NASA
Technical
Paper
3198

1992

Software Design for Automated Assembly of Truss Structures

Catherine L. Herstrom,
Carolyn Grantham,
Cheryl L. Allen,
William R. Doggett,
and Ralph W. Will
Langley Research Center
Hampton, Virginia

NASA

National Aeronautics and Space Administration
Office of Management
Scientific and Technical Information Program

Abstract

Concern over limited extravehicular and intravehicular activitiy time has increased the interest in performing in-space assembly and construction operations with automated robotic systems. A technique being considered at Langley Research Center is a supervised-autonomy approach, which can be monitored by an Earth-based supervisor that intervenes only when the automated system encounters a problem. A test-bed to support evaluation of the hardware and software requirements for supervised-autonomy assembly methods has been developed. This report describes the design of the software system necessary to support the assembly process. The system is implemented and successfully assembles and disassembles a planar tetrahedral truss structure. The software is hierarchical and supports both automated assembly operations and supervisor error-recovery procedures, including the capability to pause and reverse any operation. The software design serves as a model for the development of software for more sophisticated automated systems and as a test-bed for evaluation of new concepts and hardware components.

Introduction

A number of future space missions will require large truss structures, some of which will support functional surfaces such as antennas, reflectors, and aerobrakes. Two examples of such missions are shown in figure 1. Figure 1(a) is an astronomical observatory, and figure 1(b) is a Mars mission vehicle with a truss-supported aerobrake. Considerable effort has been expended during the past 10 years toward establishing a capability of assembling large space structures on orbit (refs. 1 and 2). A shuttle flight experiment of a large truss structure (ref. 3) and recent truss-supported reflector designs (ref. 4) are aimed at astronaut assembly. However, current concern over limited astronaut EVA (extravehicular activity) and IVA (intravehicular activity) time (ref. 5) has increased the interest in performing in-space assembly and construction with automated, robotic systems. One particularly attractive alternative utilizes the operator as a supervisor or system monitor, called upon only when the robotic system requires intervention or assistance. This mode of operation, known as supervised autonomy, eliminates planned EVA for construction and reduces IVA. Supervised autonomy has the advantage that it can be performed from any location, including the ground, since it does not require the performance of time-critical active functions by the operator.

To date, very little effort has been directed toward the development of automated robotic methods for large truss structures. Langley Research Center (LaRC) has developed a unique facility to support the first detailed study of automated structural assembly (ref. 6). The interdisciplinary effort focuses on gaining practical experience in the automated assembly of large, generic, truss-structure hardware designed for robotic operations.

The objective of this report is to describe the requirements and design of the software that performs the automated assembly of the truss structure and to discuss the interface and interaction between the software program, the system hardware, and the operator. An initial version of the automated assembly system has been developed and is currently operational. Considerable experience has been accumulated in the assembly and disassembly of a 102-member tetrahedral truss structure (refs. 6 to 8). The assembly system components are described, and a narrative of the assembly process is given, to serve as a basis for the description of the software and its functions. The actual implementation of the design is discussed in appendix A. Finally, an evaluation of the software system operation and experience is presented. The purpose of the evaluation is to discuss the success of the design in satisfying the system requirements. A glossary of terms relative to the subject matter discussed in this paper is contained in appendix B.

Symbols and Abbreviations

AP	approach point
AP_CAN	canister approach point
CAP	capture locations (CAP1, CAP2)
CLOSE, LOCK, EXTEND	individual actuator commands
GP	grasp point
GP_CAN	canister grasp point

INSTALL, REMOVE, ACQUIRE, PROP	assembly functions
IP	transition point
NASREM	NASA/NBS standard reference model
Pitch, Roll, Yaw	pitch, roll, and yaw orientations of robot arm
R	radius
REM	removal locations (REM1, REM2)
STORAGE	storage-tray grasp point
STORAGE_AP	approach point to storage-tray canister
TRAY	working-tray grasp point
TRAY_AP	approach point to working-tray canister
TRIPOD	capture point for pyramid installation
X, Y, Z	x, y, and z positions of robot arm
x, y, z	coordinate locations along x-, y-, and z-axes
θ	angle of rotation, deg

Assembly Facility Hardware

Figure 2(a) is a schematic of the automated assembly facility, and figure 2(b) is a photograph of the actual hardware system. The facility consists of a robot arm, a motion base, a truss, an end effector, and strut storage trays. It uses commercially available equipment so that it can be easily modified. The hardware system is a ground-based research tool designed to permit evaluation of assembly techniques, strut joining and end-effector components, computer software architecture, and operator interface requirements that are necessary for automated in-space operations. A more complete description of the facility hardware and performance characteristics is given in references 9 to 11.

Robot Arm

The robot arm is a six-degree-of-freedom industrial manipulator selected for its reach envelope, payload capacity, positioning repeatability, and reliability. The robot-arm computer is based on a 68000 microprocessor, and all robot-arm motions are programmed in a modified BASIC programming language supplied by the manufacturer.

Motion Base

The motion base includes a linear translational carriage and a rotating turntable. The robot arm is mounted on the carriage, which has approximately 20 ft of travel in both the x and y directions, with a positioning accuracy of 0.002 in. The truss structure is assembled on a rotating turntable capable of six revolutions of travel and a positioning accuracy of 0.01 in. at a radial distance of 20 ft (0.0024°). Motion-base drive motors on the three axes are commanded by an Intel 80286 microprocessor-based indexer.

Truss

A planar tetrahedral truss, such as the model shown in figure 3, was selected for initial assembly studies because it is representative of the type of truss structures required for large antennas, reflectors, and aerobrakes. The truss is specifically designed for automated assembly and includes regular hexagonal rings. Core struts are those that connect the top face to the bottom face. All struts are nominally 6.6 ft long and 1 in. in diameter. The complete structure has 102 struts and 31 nodes. Assembly begins by connecting struts to three nodes that are premounted on the motion-base turntable.

The truss node and joint connectors are shown in figure 4. Two joint connectors are bonded to a graphite-epoxy tube to form a strut. The joint has a connector section which, during assembly, is inserted into a node-mounted receptacle. A locking nut is turned by the end effector to draw the connector plunger toward the connector face of the strut, securing the joint. The alignment and grasp adapter is used to grip the strut and align it precisely with the end effector.

End Effector

The end effector is a specialized tool mounted on the robot arm that performs all strut installation and removal operations. Figure 5(a) is a schematic of the end effector, and figure 5(b) is a photograph of the end effector and its components. The strut is grasped by a set of strut holders that close around the alignment and grasp adapters (figs. 4 and 5) that are bonded to the strut tube. The strut holders are mounted on a platform that is extended for insertion of the strut into the node receptacles. A strut is installed into the truss by moving the end effector to a position where the receptacle fingers (fig. 5(b)) are able to grapple the node receptacle. Actuators close the fingers around the node receptacle and passively align the end effector and strut with the node receptacle. After alignment, the platform is extended and inserts both joint connectors into the receptacles, where they are held while the locking nuts are tightened with nut drivers on the end effector. The strut holders are unlatched and the platform is retracted. The receptacle fingers are then opened to release the structure.

All end-effector components and actuators are equipped with simple sensors, such as microswitches and linear potentiometers, so that the computer program can monitor the operations and alert the operator if a problem occurs. Small video cameras are mounted on each end of the end effector to permit operator monitoring of component functions.

A six-axis force-torque sensor (FTS) is mounted on the wrist of the robot arm to measure forces and moments acting on the end effector. The output of the FTS is used to command small robot-arm movements in a direction that will "zero" the measured forces and moments. This movement is used to reduce the loads on the end effector and to enable the end-effector components to operate freely.

Trays

The truss struts are stored in nine trays, which are stacked in the working canister directly behind the robot arm (figs. 2(a) and 2(b)). Empty trays are transferred by picking them up with the end effector and moving them to the storage canister, which is located on one side of the robot arm. The struts are removed from the tray by positioning the end effector over the strut, extending the platform so that the strut holders contact the alignment and grasp adapters, and latching the strut to the end effector. The platform is then retracted to withdraw the strut from the tray. Each tray has cylindrical handles on both ends; these handles are fitted with positioning and alignment adapters, which allow the end effector to pick up empty trays from the working canister and transfer them to the storage canister.

Assembly Process

The assembly process begins when the end effector acquires the first strut from the top tray in the working canister. Once the strut is acquired, the motion base is positioned so that the robot arm can connect the strut to the structure. The robot arm, moving through a sequence of predetermined points, positions the strut at its point of installation or grasp point. The end effector then inserts and locks the strut into place. Finally, the robot arm returns to the working canister to retrieve another strut. This basic operational sequence is followed for the installation of all struts. Each part of the sequence is detailed in the sections that follow.

Acquiring a Strut From the Tray

Each strut has a preassigned tray number and a slot location. The end effector is positioned at the canister approach point (a predefined point at the top of the working canister), which is directly over the desired strut in the tray. Receptacle fingers are closed to prevent collisions with preattached nodes on adjacent struts remaining in the tray. The end effector is lowered to the canister grasp point (the level of the tray containing the strut), so that extending the platform causes the strut holders to lightly contact the strut alignment and grasp adapters. When the platform extends, the force-torque algorithm balances the forces and moments acting on the end effector while slowly applying a maximum of 20 lbf in the downward direction to close the strut holders. After the strut holders are latched, the forces and torques are rebalanced. The platform is then retracted, and the strut is lifted from the working canister. From the working canister grasp point, the strut is carried to the canister approach point, where the receptacle fingers are opened in preparation for the installation operation.

Motion-Base Moves

Associated with the installation of each strut are the carriage and turntable positions (x, y, and θ) required for installation. The current carriage and turntable positions, the required carriage and turntable positions for the strut being installed, and the status of the structural assembly are used to determine if the carriage and/or robot arm will collide with any struts that are currently assembled. The motion-base repositioning commands can be performed in any order. All motion-base moves are performed with the robot arm positioned at the canister approach point to minimize the distance the robot arm extends toward the truss; this positioning reduces chances for collision. The motion-base collision-avoidance algorithm is described in detail in the section "Motion-Base and Collision-Avoidance Design."

Robot-Arm Paths

The robot arm traverses a predetermined path to deliver the strut to the proper location and orientation in the structure. There are three strut installation cases: direct, capture sequence, and pyramid completion. For direct installation, the end effector and strut are carried directly to the grasp point, a predetermined location where the strut can be installed into the structure. Direct installation entails either the insertion of a strut between two fixed nodes already in the structure or the installation of a strut with a preattached node at one end. For struts with preattached nodes, the end effector only operates the receptacle fingers and locking component at one end and leaves the strut-node combination cantilevered from the fixed node to which it is installed in the structure. The installation of a strut with a preattached node creates the capture-sequence installation, which requires the end effector to install a strut between the free end of a cantilevered strut (deflected by gravity) and another node in the structure. For this case, the end effector must be positioned so that the receptacle fingers on one end grasp and capture the cantilevered node. The robot arm is then moved so that the receptacle fingers on the opposite end can grasp the node in the structure and so that both ends of the strut can be inserted and locked into place.

The pyramid-completion installation case performs the installation of a third strut into a pyramid substructure. This installation is similar to the capture-sequence installation, except that the node being captured already connects two struts. For the pyramid-completion installation, the deflections due to gravity are not as large as in the capture-sequence installation. The robot arm is again moved to the grasp point after node capture of the two connected struts where the strut is inserted; this move completes the pyramid configuration.

In addition to the three installation cases, there are two removal cases that are necessary for disassembly: free and direct. The free removal case involves cantilevered struts with preattached nodes that are deflected as a result of gravity. The robot arm must move to a predetermined point and close the receptacle fingers to capture the cantilevered end. It then continues to a second predetermined point to avoid node receptacles of installed struts before proceeding to the grasp point, where the strut is removed from the structure. The direct removal case applies to all other struts. The robot arm traverses a straight path directly to the grasp point. The end effector receives commands during the path sequence to perform tasks such as closing receptacle fingers to capture nodes at the proper locations.

End-Effector Operations

When the robot arm reaches the grasp point for the strut, control is transferred to the end effector. A strut installation includes closing the receptacle fingers on the node receptacles, extending the platform to insert the strut into the receptacles, locking the strut into place, unlatching the strut from the end effector, retracting the platform, and opening the receptacle fingers. Sensors are monitored after each step, and the sequence does not proceed unless the operation is successful.

System Software Requirements

The automated assembly system software was developed to support projected assembly system requirements. These requirements were generated by an interdisciplinary group of hardware designers, programmers, engineers, and prospective users of the system. The participation of a wide range of disciplines resulted in a software design that has not changed appreciably during the evolution of the system. These system requirements are discussed further in the following section, and the requirements for the three devices—motion base, robot arm, and end effector—are discussed in subsequent sections.

Overall Requirements

The overall system requirements are as follows: (1) to assemble and disassemble the tetrahedral truss in an automated mode; (2) to provide sufficient information displays and control capability to support a supervised autonomy mode of operation; (3) to interface with advanced systems, such as planners; and (4) to accommodate assembly system hardware and procedural upgrades.

The requirement to provide the capability for a fully automated assembly and disassembly established the need to know the predetermined conditions that direct the assembly process, the current state of all system hardware devices, and the current state and location of every strut in the truss structure. Predetermined conditions include the geometry of the structure, path sequences, strut storage information, motion-base moves for strut installation, and potential obstructions during motion-base moves. The software must include algorithms and procedures for gravity-deflected strut capture, motion-base collision avoidance, and error recovery. When performing the assembly task, the software must command

and sequence the motion base, robot arm, and end-effector hardware, and must provide interfaces for the supervisor. Because each of the system hardware devices is an independent subsystem that must be coordinated during assembly operations, the software design must accommodate a distributed architecture to provide local device component control.

The software requirements are driven by a need for a user to monitor and effectively manage the operation of the automated system. The role of the system software user and the user interface is therefore defined as follows:

1. The user is considered to be a supervisor because system operation is primarily in an automated mode.
2. Supervisor interaction is required only for error recovery.
3. Supervisor monitoring is supported with as much task and status information as possible. This information must be clear and concise.
4. The supervisor may intervene at any time to change tasks or request information. This intervention includes pausing the automated sequences to look at video displays or assembly details before either resuming or reversing the task.
5. The supervisor has override capability over all automated functions.
6. The supervisor is not responsible for data and status updates resulting from commanded actions. These updates occur automatically.
7. A secondary manual or checkout mode allows the supervisor access to all levels of commands and data so that all automated functions can be duplicated and analyzed. System status checks are performed prior to execution of all supervisor commands to avoid damaging actions. Access to the lower command levels is restricted to experienced or authorized supervisors.
8. Three modes of supervisor input are required: keyboard, command file, and assembly-sequence file. The keyboard mode requires the supervisor to enter each command manually. The command-file mode alleviates some typing by allowing the supervisor to create a file of the actual commands that would be entered interactively. The command-file execution should parallel the performance of the system in the keyboard mode. The assembly-sequence file is a higher level command file. It contains general assembly and disassembly sequences, including an ordered list of the struts to be installed or removed. The system translates the assembly-sequence file into a command file of the actual system commands. The system allows the supervisor to perform online creation, modification, and error recovery of the command- and assembly-sequence files.

The third overall requirement is the ability to interface with advanced systems. Under current consideration are knowledge-based, expert system control of assembly functions, path-planning tools for the robot arm, and machine vision to provide robust system operation.

The software system must accommodate assembly system hardware, computer hardware, and procedural upgrades that result from operational experience. One procedural upgrade that became apparent during the development process was the need to reverse the assembly process after a pause or unresolved error. This capability improves supervisor confidence in the automated system operations and provides a powerful error-recovery technique. When an error cannot be corrected, the system automatically initiates a reverse sequence of commands and relieves the supervisor of having to remember the proper sequence. This reverse sequence imposes a significant burden on the software, however, because the reverse sequence is not necessarily the exact opposite of the forward sequence. An example of the ability to accommodate new hardware involves the incorporation of new end effectors for additional assembly tasks and advanced operations.

Motion-Base Requirements

The motion base must position the carriage and rotating turntable to the correct x, y, and θ positions. The x, y, and θ positions are expressed as either absolute locations or moves relative to the current position. The x, y, and θ moves may execute in any order, and each move is verified before the next move is begun. The motion base should be able to move to predefined locations or receive a direct move instruction from the supervisor. Before any movement of the motion bases, collision-avoidance logic must determine the order of moves that will keep the motion base from hitting the assembled struts.

A pause option for the motion base includes the ability to manually adjust the current position. When reversing the motion base, the forward sequence is retraced.

Robot-Arm Requirements

The robot arm is required to traverse predetermined paths for strut installation and removal, for moving the trays to and from the storage canister, and for changing the end effector. The robot-arm program must be able to access the end-effector commands directly to avoid obstacles and perform capture tasks.

For the strut paths, the robot arm is required to automatically move sequentially in either direction through a series of intermediate positions that depend on the strut installation position in the truss. There are 19 unique paths used during the assembly of the truss structure. The software must be able to select the correct path for each strut, including the capture of gravity-deflected cantilevered struts.

The robot-arm reverse is not always the opposite of the forward sequence for the strut paths. Details of the reverse are discussed in the section "Robot-Arm Path Design." The reverse sequences for tray operations and end-effector changes are exactly opposite of their forward sequences. As with the motion base, the pause option includes the capability for the supervisor to adjust the robot-arm position.

End-Effector Requirements

The end-effector software must be able to generate sequences of actuator commands to perform four basic assembly functions: install, remove, acquire, and drop. The system must be able to access the actuator commands, monitor sensor outputs, and perform sensor conflict checking after each actuator command is executed. The software design must be able to support various end effectors, such as the addition of a panel end effector for future assembly operations.

End-effector error conditions detected by the sensors are displayed for the supervisor. The supervisor has the option to manipulate the end-effector actuators directly, reposition the robot arm to permit the actuator to function properly, continue execution when the error is not deemed serious enough to warrant action, or abort error correction and allow the system to reverse. For the end-effector functions, the reverse is not always the exact opposite of the forward sequence. The error-recovery software provides the option of executing automatically or manually. The end-effector software must provide direct access to the robot-arm commands to reposition the robot arm or to balance the forces and torques acting on the end effector.

System Software Design

Although the assembly system is intended to operate in a fully automated mode, it is imperative that the supervisor be provided with appropriate internal information and have sufficient command access and authority to deal with assembly problems. For this reason, the automated assembly system software design is approached primarily from the supervisor's viewpoint. A command hierarchy makes the control process simple and orderly. The result is a modular software structure that coincides with the hierarchical nature of automated operations. The following sections provide the details of the software design. Appendix A provides some insight into the actual implementation of the design.

Design Overview

Design Layout

Figure 6 shows the design layout of the automated assembly program. It comprises four basic levels: administrative, assembly, device, and component. Because of the natural hierarchy of the assembly process, a top-down design philosophy is used; this philosophy causes the highest level commands to appear first and successively decomposes to the lowest-level component commands. The software design process is based upon the assembly sequence described previously and the requirement that the supervisor have access to all levels of detail.

The administrative level is involved with the preliminary setup of the system. It allows the supervisor to examine and modify data and system options. The command and assembly files can be selected, created, and modified. Also, the supervisor gains access to the lower levels of the system through the administrative level.

The assembly level reflects the automated aspect of the system. At this level, the software manages all the devices, data verification, and error recovery. Command operations at this level for assembly and disassembly of the truss are all automated. This level interfaces with a proposed automated task sequence planner. The standard operating mode occurs at the administrative and assembly levels.

The device level gives the supervisor access to each individual device and to the functions the device performs. To obtain and install a strut requires action by three separate devices: the motion base, the robot arm, and the end effector. Decomposition of the commands at the assembly level results in a sequential list of device-level commands. The functions associated with each device are taken directly from the requirements. Access to this level requires more expertise on the part of the supervisor and involves less automatic checking by the software.

The component level reflects the hardware capability of the current system. Each of the device commands, such as the end-effector install command (INSTALL) decomposes into individual actuator commands (e.g., CLOSE, LOCK, EXTEND) which are the basic tasks performed by the hardware. Sensor checking and verification occurs after execution of each component command. This level is dependent on the specific devices used and could change if the hardware changes an important aspect to consider in the software design and implementation.

Menu Interface

A menu-driven, rather than a command-driven, interface is used in an effort to reduce the number of commands at each level and the amount of internal system information presented to the supervisor. The menu-driven command structure also accommodates relatively inexperienced supervisors. Figure 7 shows the basic menu layout for the system; the layout was derived directly from the design in figure 6. The menus reflect the actions required to control the hardware and assembly process.

Each box represents a menu on the supervisor's display. Menu selections can be designated by either the number or the first unique characters of the command. The lines between boxes indicate how a supervisor traverses the various levels of the system. Every menu contains "help" to aid inexperienced supervisors by providing information about each selection. As a selection is made, the item is highlighted. The menus are overlapped on the screen as they are selected (fig. 8) to provide the supervisor with information from every level. Everything entered by the supervisor is recorded in a journal file that is available for post-test analysis.

Command Decomposition

The main menu (figs. 7 and 8) displays the four major components of the system. Selection 1 (System configuration) allows the system configuration parameters and variable status to be displayed and modified. Selection 2 (Auto build) initiates automated assembly according to a predetermined assembly sequence contained in a predefined assembly-sequence file. Selection 3 (Assembly functions) allows access to the manual command mode, which provides the supervisor with command capability at all levels of the automated system. Selection 4 (File manipulation) permits selection and editing of an automated assembly-sequence file or a command file; these files are discussed in the following section.

Selection 3 reveals subsequent hierarchical menus in which higher level menu commands are composites of lower level menu commands. The lowest level menus are the component-oriented commands that are directly associated with the hardware of the system. All commands incorporate internal, automatic checking to protect the hardware from supervisor-controlled commands that could result in hardware damage. As the supervisor works down the menu hierarchy, control and responsibility shifts from the automated system to the supervisor. The lower level menus rely on supervisor expertise; therefore, many of the lowest menu levels and some error menu selections are password-protected or have hidden menu options.

The composite commands are higher level menu entries that initiate a sequence of commands to perform the selected task. Figure 9 illustrates composite command Fetch and connect. As each command is executed, the associated menu is displayed and highlighted. This layered menu presentation allows the supervisor to monitor the sequence of hierarchical commands and provides a trace to aid in error recovery.

An error-recovery menu is displayed to the supervisor when sensor checks indicate that a system component did not function properly. The system will not proceed until the problem is resolved. If a problem cannot be corrected, error information is passed back through the system hierarchy and causes the commanded actions to reverse their task.

Supervisor Input Modes

There are three modes of supervisor input: direct keyboard, command file, and assembly-sequence file, as defined in the requirements. The keyboard input mode requires the supervisor to enter each menu selection from the keyboard. The command-file mode allows the supervisor to create a text file of the commands as they would be entered in the keyboard input mode. The system obtains its input from the file, so that the supervisor is freed for monitoring. This freedom is particularly helpful for repetitive tasks. A command file is executed through selection 4 (File manipulation) from the main menu. This menu also allows the supervisor to create a command file (Build command file) and modify an existing command file (Edit command file) without exiting the program.

Limited on-line correction of the command file is available when an illegal command is encountered. Execution is suspended and the command-file error menu is displayed, as shown in figure 10. The current line in the command file, the command containing the error, the next command, and a list of actions available to the supervisor are displayed. In the example shown, the current command file contains an incorrect command. The supervisor should pick selection 1 (Correct current command) and will be prompted to enter a new command. After correcting the command file, the supervisor can execute the file two ways. Selection 2 (Re-execute current command) will execute the corrected command, increment the command file to the next line, and return to the command-file error menu. This option allows the supervisor to insert commands and execute them one by one. The second way to initiate execution is by picking selection 3 (Continue execution with current command). This option starts executing at the corrected command, continues execution from this point, and exits the command-file error menu. This option is similar to selection 4 (Execute the next command and continue), but this option begins at the next command and skips execution of the current command.

In situations where many commands need to be modified, it may be more efficient to abort the command-file execution mode and edit the command file. The command file may be edited without exiting the program by selecting main menu item 4 (File manipulation).

The third input mode, an assembly-sequence file, executes like the command-file mode, but the format is independent of the actual commands entered. The format is simplified and the software converts the file into the commands required by the system. The assembly-sequence file format is as follows:

```
Assemble
    strutname_a
    strutname_b
    .
    .
Disassemble
    strutname_c
    strutname_d
    .
    .
End
```

The supervisor has the option of creating and modifying these files without exiting the system.

Robot-Arm Path Design

The robot arm has three tasks to perform: (1) traverse strut paths for installation and removal; (2) transfer trays between the working and storage canister; and (3) change the end effector. Transferring trays and changing the end effector are fairly straightforward tasks. Traversing the strut path is more complex because of the intricate orientations necessary to locate the strut in the structure without interference from previously installed struts. Robot-arm tasks are detailed in the following sections.

Logic of Strut Path State

The robot-arm path from the strut storage canister to the structure and return has been divided into segments or path states. The exact path traversed depends upon the current strut location in the structure. The state is the current coordinate location of the robot arm (X, Y, Z, Roll, Pitch, Yaw). The states defined for this study are illustrated in figure 11 (GP_CAN, AP_CAN, IP, AP, and GP). This illustration typifies the simplest sequence of moves required to carry a strut between the canister and the structure.

The robot-arm rest position and the point at which it begins a strut retrieval is located immediately above the canister and is designated the canister approach point, AP_CAN. The strut is picked up at the canister grasp point, GP_CAN, and carried back to AP_CAN. A transition point, IP, is passed through before the strut is carried to the structure approach point, AP. The transition point is where a transition occurs from a canister-oriented path, which involves a tray and slot number, to an installation-oriented path that is dependent on the strut location in the structure. The approach point is approximately 4 in. from the grasp point at the structure, GP, where the strut is actually installed.

Figure 12 is a complete diagram of the robot-arm state paths, including capture operations. The figure indicates the strut installation and removal cases, which determine the various paths, as well as conditions for performing end-effector actions. Conditional states, denoted by dashed boxes, from AP to GP are special cases required for various strut cantilever conditions. The conditional state positions enable the robot arm to capture cantilevered struts and avoid collisions with node receptacles while lining up the strut at its location in the structure. The states are represented by solid boxes, and the arrows between the states represent transitions between states. End-effector receptacle finger actions, as shown by the ovals, are required at various points along the path sequence. The robot arm passes sequentially from one state to the next when moving between the canister and the structure. The arrowheads indicate the directions allowed between states, conditional states, and end-effector actions. The only time the system can terminate between states is when a hardware failure occurs.

When determining the required path from state to state, the paths with conditions are considered first. The unconditioned path is taken only when none of the conditions for capture operations are met. Cases exist for which the reverse conditions do not mirror the forward path. When the robot arm is following a path through a sequence and a reverse is initiated, the arrowheads are followed. If no arrowhead points in the reverse direction along the path, a new path is determined by continuing until a state or conditional state is reached that contains an arrowhead in the reverse direction.

The path between AP and GP is dependent on the installation or removal case, as specified by end conditions of the strut. Struts that attach to fixed nodes and those that are attached at only one end proceed directly from AP to GP (direct installation case). Struts that must capture previously installed cantilevered struts move through points CAP1 and CAP2 to perform the capture maneuver (capture-sequence installation case). Receptacle fingers are closed at CAP1 to capture the node, and for those installations that require the capture of two nodes, receptacle fingers on the opposite end of the end effector are closed at CAP2. The struts that capture only one node also travel to a CAP2 point but do not close the fingers before proceeding to GP. Maneuvers that capture the node of a connected pair of cantilevered struts perform the capture at a point called TRIPOD (pyramid installation case). The newly connected strut completes a pyramid configuration.

The struts that are attached only at one end are left cantilevered, and gravity causes them to sag when they are released by the robot arm. The robot arm must move to the deflected points before releasing these struts to avoid entangling the receptacle fingers on the node receptacle. The same is true when removing the strut. A set of points designated REM1 and REM2 are used for cantilevered struts (free-removal case). The consistency of strut deflections makes it possible to use predetermined points for the capture and remove (CAP, REM) locations. The gravity-induced strut deflections and predetermined points are not viable in space applications. Deflections in the zero-gravity environment are smaller, but are in random directions; this randomness dictates the use of sensors such as machine vision. However, the concept of robot path segments for retrieving combinations of struts and nodes and for avoiding previously installed node receptacles is still valid.

The supervisor may interrupt a move at any point. A supervisor pause stops the robot arm immediately and displays a pause menu on the screen (fig. 13). The supervisor can then proceed from the point of interruption, adjust the robot-arm position, or return to the originating state. The supervisor must be aware that this originating state is not the previous state in the path, but the originating state of the sequence. For example, if the robot arm is currently at AP_CAN and commanded to move to GP, a pause-reverse requested at AP causes it to return to AP_CAN. The robot-arm motion may be paused and reversed as many times as the supervisor desires, this process acts as a toggle to change the robot-arm direction.

Logic of Tray Path State

The tray path states are less complicated than the strut path states. Figure 14 illustrates the four states for tray storage (TRAY, TRAY_AP, STORAGE_AP, STORAGE). To move a tray from the working-tray canister to the storage-tray canister, the robot arm must first move from the approach point to the working-tray canister (TRAY_AP). The robot arm then moves down to the tray grasp point (TRAY) and picks up the tray exactly as if it were acquiring a strut. The tray is carried back to the approach point (TRAY_AP) and then to the storage-canister approach point (STORAGE_AP), which is located at the top of the storage canister. The tray is then moved down to the storage grasp point (STORAGE) and is released in the same manner as a strut being placed in the canister. After releasing the tray, the robot arm retraces its path back to the working-canister approach point and is ready to resume strut installation. To retrieve a tray from the storage canister, the same path is followed, except that the pickup is performed in the storage canister and the release in the working canister. There are only two tray operations: storage and retrieval. Once an operation is selected, execution proceeds sequentially with no decision points.

Logic of End-Effector Change

The path logic followed for changing the end effector is the same as that for moving the trays. The end-effector storage approach point, the actual storage grasp point, the retrieval approach point, and the retrieval point are predetermined locations.

The robot arm first proceeds to the storage approach point and then to the storage grasp point. After disengaging the end effector, the robot arm returns to the storage approach point and then proceeds to the approach point of the end effector to be retrieved. The robot arm then continues to the retrieval grasp point, attaches a new end effector, and returns to the retrieval approach point.

Motion-Base and Collision-Avoidance Design

The current motion-base controller commands move in a sequential manner, one axis at a time. The x, y, and θ positions associated with a particular strut installation are either obtained from predefined locations or input by the supervisor. Before any motion-base repositioning is initiated, a collision-avoidance algorithm is executed to determine the order of sequential moves that will prevent collisions with the structure. A new axis move is initiated only after the previous move is complete.

The supervisor may intervene and pause at any time during the move sequence. In the paused condition, the options are to continue, adjust, or reverse. The adjust accepts an intermediate set of positions from the supervisor. When reverse is selected, a retrace of the forward sequence is executed.

The collision-avoidance logic determines the order in which carriage moves must be performed to prevent the carriage and robot arm from colliding with any part of the structure already assembled on the turntable. Collisions can occur during x-axis moves when traveling toward the structure, and during carriage y-axis moves and turntable rotations if the carriage is positioned too close to the existing structure. For x-axis moves, collisions occur between the carriage and the installed bottom-face struts. For y-axis and turnable moves, the elbow of the robot arm and the handles of the empty trays that protrude from the storage canister are the two potential collision points. (See fig. 2(a).) A set of tests are used to examine each of the two potential collision points. Only the installed core struts (those which connect the top and bottom faces of the truss structure) are considered for collision because they are at the same height as the elbow and tray handles. All calculations for collision avoidance are performed on-line prior to the installation of each strut.

Figure 15 defines the nomenclature to be used in the discussion of the collision-avoidance problem. Associated with each strut is the point where a collision can occur (strut end point) and the angle θ_{strut} between the radius of this point R_{strut} and the turntable x-axis reference line. Collision avoidance is discussed for an x-axis move, a y-axis move, a turntable rotation, and a combination of y-axis move and turntable rotation. The y-axis move and turntable rotation algorithms are applied twice—to check for potential collision problems with the robot-arm elbow and then with the tray handles. The following text outlines the algorithm used for collisions that may occur with the elbow.

Logic for x-axis move. The x-axis carriage moves are not a primary concern in collision avoidance because of the structural configuration of Automated Structures Assembly Laboratory. The use of the predefined points guarantees the proper clearances. The x-axis move algorithm is only necessary when the supervisor has requested direct access to the motion base and has thereby manually entered the coordinates. Collision avoidance is performed for x-axis moves that position the carriage closer to the structure. An x-axis collision occurs when any installed bottom-face strut intersects the new carriage position. When this happens, the move is illegal and not performed by the system.

Logic for y-axis move. Figure 16 illustrates the collision-avoidance algorithm for a y-axis carriage move. The radius of a potentially obstructing core strut R_{obs} is the distance from the center of the turntable to the end of the strut farthest from the turntable. This radius is represented by a line extending from the turntable center. The desired or next position of the carriage is depicted in the figure by dashed lines.

Two tests are performed to identify potential collisions. In the first test, the smallest absolute angle (fig. 16) is computed between the x-axis reference line and the obstructing strut θ_{obs}, the current robot-arm radius θ_{start}, or the desired robot-arm radius θ_{end}. When the angle of the strut radius lies outside the robot-arm angles, the move can be performed (case 1). When θ_{obs} lies between the two angles formed by the robot-arm radii (cases 2 and 3), a collision may occur and a second test must be performed.

In the second test, a new carriage radius R_{carr} is computed and the carriage location is assumed to be at the point of the obstruction. The carriage radius is depicted in the figure by the bracket and the radius of the obstruction R_{obs} is the length to the dot. This new radius is then compared with the strut radius of the potentially obstructing strut. If the strut radius is less than the carriage radius (case 2), the move can proceed. When the strut radius is greater than the carriage radius, corrective action must be taken (case 3). Before the move can proceed, the carriage must be moved back in the x direction. The distance of this move, with a safety factor, is computed from the length of the strut radius and the angle of the obstruction as follows:

$$x_{\text{distance}} = R_{\text{obs}} \cos\left(\theta_{\text{obs}}\right)$$

Logic for turntable rotation. Figure 17 illustrates the collision-avoidance algorithm for a turntable rotation. In case 1, the strut radius R_{obs} is compared with the carriage radius R_{carr}. If the strut radius is greater than the carriage radius, the turntable rotation direction is examined, as in case 2. The angles are calculated for θ_{carr}, θ_{start}, and θ_{end}. The angles are compared, and the turntable can be rotated if $\theta_{\text{carr}} > \theta_{\text{end}}$ and $\theta_{\text{end}} > \theta_{\text{start}}$. Otherwise, case 3 governs, and the carriage must retreat in the x direction before performing the move. The distance of the x-axis move is computed in the same manner as that for the y-axis move as follows:

$$x_{\text{distance}} = R_{\text{start}} \cos\left(\theta_{\text{carr}}\right)$$

Logic for combination y-axis and turntable rotation. A scenario is assumed in which the y move occurs before the turntable rotation. If no retreat in the x direction is necessary, the move is completed; otherwise, a turntable rotation followed by a y-axis move is considered. If this combination proves collision free, it is executed. When a retreat in the x direction is necessary for both combinations, the combination is performed that produces the smallest move in the x direction.

End-Effector Design

Initially, the end-effector task was to generate actuator command sequences for the four assembly functions (INSTALL, REMOVE, ACQUIRE, DROP) and to monitor sensor output. However, operational experience established a need to provide effective error recovery. Error-recovery techniques were developed as the error sources were identified during actual assembly operations. The need for the pause and reverse capability for the supervisor, and the ability to reverse following an unresolved error, significantly complicated the sequencing algorithm. Much more software was required to implement these functions than was originally anticipated.

End-Effector Component Commands

The end-effector component commands control the actuators and are the lowest level accessible to the supervisor. The end-effector hardware is shown in figure 5(b). The end-effector component commands and a brief explanation of each task follow:

OPEN/CLOSE	Commands the receptacle fingers to open or close
EXTEND/RETRACT	Commands a pneumatically actuated platform to be extended or retracted pushing or pulling a strut
LOCK/UNLOCK	Secures or releases the strut to or from the structure
LATCH/UNLATCH	Commands a pair of strut holders to close or open around the alignment and grasp adapters located on the strut

There is a set of receptacle fingers on each end of the end effector and a locking nut on each end of the strut. Therefore, the OPEN/CLOSE and LOCK/UNLOCK commands can be executed individually for the left side and the right side. The end-effector platforms and strut holders on both ends of the end effector work simultaneously.

Each of these elemental component commands implies a self-contained task that is performed by the end-effector software. The component sensors are checked prior to issuing actuator commands, and the software issues the command when the status is not in the desired state. Assembly proceeds if sensor checking indicates that the operation was successful; otherwise, an error is returned and the software is suspended at this point until the error is resolved.

End-Effector Functions

The operational sequences for the end-effector assembly functions are described in this section. These functions represent the device-level end-effector commands and are made up of a sequence of component commands. The functions and a brief explanation of each task follow:

ACQUIRE	Picks up a strut from the tray and retains it in the end effector
DROP	Puts a strut into the tray and releases it from the end effector

INSTALL	Inserts and locks a strut into the structure
REMOVE	Unlocks a strut and removes it from the structure

The component commands are shown in figures 18(a) to 18(d) for the device-level commands discussed in the preceding paragraph. Figure 18(a) shows the sequence for the ACQUIRE command. The column on the left lists the sequence of end-effector component commands and robot-arm commands that perform the function. The right column, reading up, contains the sequence to reverse the ACQUIRE command. The reverse sequence is not the opposite of the forward sequence.

The first component command issued in the ACQUIRE sequence is to UNLATCH the strut grippers as a precautionary or safety feature. Next, the platform EXTEND is executed and is followed by the automatic force-torque algorithm (BALANCE FTS) to accurately align the end effector with the strut before the strut holders LATCH. A second BALANCE is executed after the LATCH, so any alignment errors that occur during the LATCH are relieved and the strut pulls smoothly from the canister during the platform RETRACT. At this point, the ACQUIRE sequence is complete and the status is updated to reflect the fact that the end effector is now carrying the strut.

A pause capability is available for all end-effector functions and may be initiated at any point in the sequence. When the supervisor pauses, the pause menu is displayed; at this point the supervisor can resume operation by either continuing with the next step or initiating the reverse sequence. The sequence may be repeatedly reversed.

The implementation of the other three end-effector functions (DROP, INSTALL, and REMOVE) is similar to the ACQUIRE implementation, and their command sequences are shown in figures 18(b) to 18(d).

Error Recovery

Error conditions detected by sensors are reported to the supervisor for selection of error-recovery actions. Two types of actions are possible—the end-effector actuators can be manipulated, or the robot arm can be repositioned to permit the component to function properly. The robot-arm motions are either supervisor-controlled adjustments in robot-arm position or products of the automatic force-torque algorithm. All error-recovery actions are selections from menus specific to each particular error, with the exception of the the force-torque algorithm, which is automatically invoked by receptacle-finger closure errors.

An error menu is displayed whenever an end-effector component fails to function properly. Selections in each of the component error menus have been determined through experience. The error-recovery menu for the receptacle fingers (grippers) is shown in figure 19. Each error menu has an exit selection (Quit) which allows termination without correction of the component error. This exit results in an automatic reversal of the action of any end-effector function currently in progress. The error menu contains a hidden option (Go on anyway) and is only available when the supervisor uses a password. This option is selected if the supervisor considers the error to be of minimal consequence and decides to assume responsibility and continue the assembly operation. The system interprets this response as if the error were corrected.

There are three ways to exit the error-recovery routine. An automatic exit results upon successful resolution of the error. The other exit conditions (Quit and Go on anyway) are supervisor-controlled as discussed above. The software remains in the error routine until one of these conditions occurs. For the recovery options, the status of the problem component sensor is checked to determine whether the recovery action was successful. The POP command is used when the locking nut socket is not seated. The slight turn helps to align the socket with the nut. Descriptions of the recovery options are listed below:

CYCLE	Reverses the command and then reexecutes it
TOGGLE	Reverses the command that failed
LATCH ANYWAY	Latches the strut gripper, even if the grippers are not closed on a strut
UNLATCH ANYWAY	Unlatches a strut from the end effector
CW POP	Turns the nut-driver motor one quarter turn in a clockwise direction
CCW POP	Turns the nut-driver motor one quarter turn in a counterclockwise direction

DITHER ARM	Moves the robot arm through small cyclic motions in a particular direction in an attempt to jar loose a stuck component
BALANCE FTS	Reduces the loads on a component by slightly repositioning the robot arm
ADJUST	Manual repositioning of the robot arm by the supervisor

Data Content and Modification

The assembly system conditions are stored in a shared data base, which contains two basic types of information—the current status of all elements of the assembly system and structure, and the predetermined positions that are used to direct and control the robot arm and motion bases. The current status information is maintained continuously to represent the physical state of the system at any point in time and to thus ensure continuity of system operations. The status is updated automatically during test runs. The predetermined position information for the robot arm and motion base includes locations and orientations that are associated with the installation of individual struts. The predetermined position information also describes the collision-free paths that the robot arm and motion base follow between the canister and the various installation positions in the truss.

Data Description

Figure 20 illustrates the data section that is broken down into the following elements: motion-base position, strut type, robot-arm status, tray status, tray handle locations, current strut status, current motion-base position, and end-effector status.

The MOTION_BASE_POSITION record stores the x, y, and θ values (X_Car, Y_Car, and Turntable) for the predetermined motion-base locations that establish the positioning relationship between the robot arm and the truss. There are 70 unique motion-base positions for the 102-member truss. In an attempt to minimize motion-base moves, many struts are installed with the motion base situated at the same position. Also, the 120° rotational symmetry of the structure allows the x and y carriage positions to be repeated for comparable struts at three locations around the structure.

The STRUT_TYPE record contains all the data necessary to describe the installation and storage conditions for each of the 102 strut members. Each strut is identified and accessed by a unique alphanumeric designation (Name). The current location of the strut (Where) is accessed by the system before any strut operation can be initiated. The system must know if the strut is currently in its tray, installed in the structure, or held by the end effector. When a strut is selected for installation, the system refers to a list of struts (Connect_To), which defines those struts that must be installed in the truss prior to installation of the selected strut. This check is a safety feature to ensure that the required initial conditions for installation of the selected member are satisfied. The secondary reference to the location status (Where) of each strut on this list certifies that all required struts are installed. The installation position (Loc_In_Cell) identifies which of the 19 predetermined paths is to be followed to install or remove a strut. The end of a strut with a preattached node (Node_End) indicates which nut driver on the end effector must not be operated while installing the strut. If the end effector must capture another node, the end to be captured is specified (Cap_End). The end condition of the installed strut (Cantilever) is used to establish predefined modifications to the path which must occur during the capture sequence. Because of tray packing limitations, a preattached node may not be located on the correct end associated with a direct path entry. This condition is identified (Flip) and initiates a robot-arm command to rotate the strut 180° at the transition point in the strut installation path. The assigned tray and slot positions (Tray, Slot) are required to replace or insert a strut in the tray. Each state in the predefined path defines the robot-arm positions (State_Pos). The collision-avoidance algorithm requires that the end position of the core struts (X_End, Y_End) be defined for computation of potential collision conditions.

The ROBOT_STATUS record contains the current positioning point for all strut paths (State, Cond_State). The current strut in the robot arm (Strut_Now), the strut in the canister to be retrieved (Strut_Getting_Now), or the last strut that was installed or removed by the robot arm (Strut_Just_Had) are represented in this record.

The TRAY_STATUS record maintains all information pertaining to the strut storage trays. The path-state identifier (Tray_State) and the objective of the move (Tray_Mode) are used to store or retrieve a tray. The number of the tray (Current_Tray) that struts are being removed from or stored in is also maintained. The approach points to the working canister (Working_Ap) and to the storage canister (Storage_Ap) are available in this record.

The TRAY_HANDLE_LOCATIONS record contains the tray handle position in the working and storage canisters utilized by the robot arm when transferring them from one canister to the other. The positions (Storage_Loc, Working_Loc) are the sets of x, y, z, roll, pitch, and yaw needed by the robot arm.

The CURRENT_STRUT record contains information pertinent to the end effector for the strut that is currently held by the end effector. The status variables indicate whether the nut-driver sockets are seated to lock or unlock the joint connector (Left_Seat, Right_Seat) and indicate the current status, locked or unlocked, of the joint (Left_Nut, Right_Nut).

The CURRENT_MOTION_BASE_POSITION record stores the current x, y, and θ (X_Car, Y_Car, and Turntable) positions of the motion bases.

The END_EFFECTOR record maintains the current status of the various components on the end effector. The status of the receptacle fingers at each end of the end effector (Left_Receptacle_Finger, Right_Receptacle_Finger) indicates whether they are open or closed. The position of the platform (Platform) and the condition of the strut holders (Latch) are also maintained. The last data item is the location needed by the robot arm to store and retrieve the end effector (Storage_Pos).

Appendix C provides an example of the data interdependence and how the data are used by the software system to perform its functions.

System Data Modification

Data examination and modification is available through selection 1 (System configuration) on the main menu (fig. 7). This selection provides a direct method for accessing the status of any component and evaluating current conditions. Upon selection of this option, a menu displays the status of the robot arm, current strut, and end effector. If the supervisor needs to change a value, a selection of that item in the menu results in a list of possible values. When changing a value that affects other data items, the supervisor is forced by the software to change them all. For example, if the strut location is changed to the robot arm, the end-effector status data must reflect that the end effector is latched to a strut. To change the value of any data, the supervisor must enter a password. This password protects the data from haphazard modifications by inexperienced supervisors and permits complete flexibility in control of variables for system setup and testing.

Software Design Evaluation

Four complete assembly and disassembly tests of the 102-member truss structure have been conducted. The supervised autonomy mode of operation has proved effective and has allowed the supervisor to correct almost all the assembly problems from the console. The successful performance of this relatively rudimentary research prototype is encouraging for in-space assembly and construction.

The software program is a major factor in the overall system success. The software design requirements have been met, and the software hierarchical structure has remained essentially unchanged, while continuing to support system evolution, especially for error-recovery procedures and system upgrades such as the end-effector microprocessor discussed in appendix A. The hierarchical structure agrees with the NASA/NBS Standard Reference Model (NASREM) architecture (appendix D) and fits the assembly problem well. A key factor in the success of the program was a realistic representation of the system hardware and assembly procedures in data structures. This representation is difficult to achieve and requires detailed consideration of the assembly problem. The benefits of an expert system implementation (appendix A) in terms of development time and code size are apparent.

Supervisor displays that depict the hierarchical commands and assembly situation in real time adequately provide status, context, and trace information for monitoring and error recovery. No formal human-factors studies have been performed, but an excellent test-bed for evaluation studies exists. A large proportion of the assembly software is concerned with keeping the person in the loop, particularly with providing full access and control at every level.

Implementation of a distributed system architecture and a teleoperator mode of operation needs to be addressed. The assembly software is just beginning to address a distributed system architecture, but no consideration has yet been given to task interdependence and scheduling. On-line path and task planning is necessary for a truly viable in-space application to be possible. A teleoperator mode for supervisor intervention is critical for in-space error recovery, because the supervisor must have complete control over the assembly operation at each level.

Concluding Remarks

An initial version of an automated assembly system for truss structures has been developed and is currently operational. Experience gained during the assembly and disassembly of a 102-member tetrahedral truss demonstrates successful performance of the automated system and of the supervisor interface used for monitoring and intervention. Based on this experience, the software design, hierarchical structure, and internal data representation described are typical of what is required for automated operations and show promise for use in projected in-space assembly and construction projects. The software requirements and design serve as a model, as well as a test-bed, for the development of software required by more sophisticated automated systems.

The software design process emphasized the importance of defining the interface requirements and the role of the supervisor. The interface between the automated system and the supervisor provides a concise method of displaying possible command selections, access to all device levels, and current system task execution and status. The supervised-autonomy mode of operation makes system supervision from remote sites, such as the ground, feasible. This mode of operation minimizes the demand for limited astronaut resources.

Hardware test experience identified unanticipated but critical automated system capabilities, such as the need to pause and reverse the assembly process. The testing also underscored the value of a well-informed supervisor in any automated operation.

NASA Langley Research Center
Hampton, VA 23665-5225
May 19, 1992

Appendix A

Implementation

The assembly system is managed by several digital computers that are serially connected through RS-232 communication lines. The administration, assembly, and device levels (fig. 6), and the operator interface functions reside on a minicomputer and are implemented in FORTRAN. Component-level functions reside on auxiliary computers. The software design was developed independently of a computer hardware configuration and has been run on a number of different computer arrangements. All communications are passed through the minicomputer, even though functionally they might be issued directly from one machine to another. The data passed between processors are written in ASCII format. This human-readable format allows stand-alone checkout to be performed on simple terminals.

The motion base is controlled by a commercial indexer board hosted on an Intel 80286 based processor. Commands to this processor are generated by a BASIC program that serves only as a translator for the positioning commands. All the collision-avoidance calculations are performed in real time on the minicomputer.

The robot-arm motions and end-effector component commands are controlled by a BASIC program on a 68000 processor. The robot-arm processor stores data locally (all the x, y, z, roll, pitch, and yaw positions) and describes the operational position definitions and paths used for the assembly operations. This local data storage minimizes the amount of information passed between the processors.

Two major changes have been made to the initial implementation—a software language substitution and a computer hardware addition. As a result of the modularity of the design, the upgrades were easily performed. The software change entailed the development of the robot-arm, path-state logic as an expert system; this system replaced the original FORTRAN implementation. The computer upgrade that was initiated involved moving the device and component level for the new end effector to a microprocessor. The device level of the current end effector resides on the minicomputer, and the component level resides on the robot-arm processor. Both these upgrades are discussed in more detail in the following sections.

Expert System Implementation

Traditional programming languages such as FORTRAN and BASIC are not well suited for encapsulating the knowledge required for complex assembly sequences. Preliminary investigations into the application of expert system technologies to perform the decision-making portions of the software system have been very encouraging.

The Knowledge Engineering System (KES) expert system development tool was utilized in this implementation (ref. 12). Rule-based, backward-chaining techniques are applied to accomplish the decision making or inferencing. A set of antecedent/consequence (if/then) rules have been formulated which capture knowledge pertaining to the path selection for strut assembly and disassembly. These rules, along with attributes and procedures, are contained in a file known as the knowledge base. Backward chaining (goal-directed inferencing) applies deductive reasoning to the specified rules, whereby a given conclusion follows directly from a known premise.

The path from the grasp-point canister (GP_CAN) to the grasp-point (GP) is decomposed into a number of individual states. (See fig. 12.) The current location of the strut, the current location of the robot arm, the type of strut being manipulated, and the task specified by the automated system or the supervisor (via menu selection) are all factors in determining the sequence of states that make up the robot-arm path. Rules have been developed to implement the state logic shown in figure 12. These rules determine the direction of the robot-arm motion and any necessary conditional states between AP and GP. The direction of robot-arm motion is determined from the current location of the robot arm, the current status of the strut, and the task or target state entered by the supervisor. Conditional state rules are invoked when performing node capture operations between AP and GP and are primarily dependent upon strut cantilever conditions. Figure 21 contains examples of conditional state rules. Once a move has been determined, forward inferencing is initiated to build the command string, which is sent to the robot arm. The KES forward inferencing uses event-driven procedural techniques that, like conventional programming languages, were structured sequentially.

This expert-system tool provides an embedding technique for integrating expert systems with procedural language code. The procedural code is able to send, receive, and modify data from a knowledge base through the use of run-time functions and special data types. The embedding technique gives the automated assembly system access to expert-system techniques for decision making, but it leaves the existing operator interface intact. An expert-system solution to the path determination portion of the algorithm was incorporated with little difficulty by utilizing the existing menu structures and input/output (I/O) handling capabilities.

The concise representation afforded by the rule-based expert system reduced the lines of code significantly and increased the maintainability of the software. Approximately 850 lines of FORTRAN code were condensed into 20 simplified KES rules. Even during the early stages of the development, modifications and upgrades were performed rapidly. The success of the expert-system implementation has prompted the application of these techniques to other modules of the assembly-system software, such as tray handling, error handling, and collision avoidance.

End-Effector Microprocessor Implementation

All end-effector functions for the new end effector are now implemented on a microprocessor. This end-effector software logic is implemented in the "C" programming language on a Siemens SAB 80535 microprocessor. The development system selected is ANSI C compatible and includes language extensions that provide access to all processor-dependent features. The SAB 80535 microprocessor supports analog-to-digital conversion and bit I/O. The software is responsible for both sequence control and sensor monitoring for all end-effector operations. The software maintains local data that describe the status of the end-effector and sensor components on the microprocessor.

The end-effector microprocessor implements the device and component levels of the assembly software. It decomposes the assembly-oriented device commands (INSTALL, REMOVE, ACQUIRE, and DROP) into component commands and monitors the sensors. The microprocessor integrated easily into the automated assembly system as a result of the design hierarchy and modularity. By standardizing these functions, multiple end effectors can be accommodated that perform similar functions, such as the INSTALL, on different entities. Thus, end effectors that install struts and panels all look the same to the automated assembly system. The end effector on the microprocessor takes advantage of the experience gained in the baseline automated assembly operations. Reference 13 contains additional details on the end-effector microprocessor implementation and software.

Appendix B

Glossary

actuator	device that applies force to move a mechanism
backward-chaining inferencing	goal-directed approach of decision making; the pursuit of a goal may require the determination of substates, which themselves may require a subgoal solution
component	any one of the end-effector hardware mechanisms
embedding	combining conventional programming applications with an expert system to form a single executable program
expert system	a computer program that uses knowledge and reasoning techniques to solve problems that normally require the services of a human expert
knowledge base	file that contains the facts and heuristics that represent human expertise about a specific domain
knowledge-based expert system	subset of the general area of expert systems in which an expert's knowledge about a class of problems is maintained in one file (knowledge base); a separate reasoning mechanism operates on this knowledge to produce a solution
RS-232 communication line	a communications protocol for transmitting information between two computers in a serial mode (one bit at a time)
rule-based expert system	system that uses antecedent/consequence (if/then) constructs to represent knowledge
degrees of freedom	number of independent position variables that would have to be specified to locate all parts of a mechanism
supervised autonomy	a mode of system operation in which operator attention or intervention is required only when a problem has occurred that cannot be corrected by the automated system
top-down design	a methodology that begins by laying out an overall program structure and successively defining lower levels in increasing detail

Appendix C

Example of Data Accesses and Modifications

The following example shows how the data defined in figure 20 are actually referenced and used by the software. The actual menu items from figure 7 are shown in bold. Only the forward execution of the command stream is shown. No pause or reverse sequences are included. To keep the example as simple as possible, it is assumed that everything passes the appropriate tests necessary to continue execution. The strut in this example is of the direct installation case type.

Commands	Data accessed
FETCH AND CONNECT:	
Input strut name to fetch	
Check data to verify valid name	STRUT_TYPE.Name
Verify necessary struts installed	STRUT_TYPE.Connect_To
Update data	ROBOT_STATUS.Getting_Now = Strutname
FETCH:	
Verify no strut currently in robot arm	ROBOT_STATUS.Strut_Now = NONE
Verify strut location	STRUT_TYPE.Where = CANISTER
Check access to tray; do one of the following:	
1) Current tray contains strut	STRUT_TYPE.Tray = TRAY_STATUS.Current_Tray
2) Next tray contains strut	STRUT_TYPE.Tray > TRAY_STATUS.Current_Tray
ROBOT: (Move tray to storage)	
MOVE TRAYS:	
TO STORAGE:	TRAY_STATUS.Tray_Mode = STORING
TRAY APPROACH POINT	
Verify current state	TRAY_STATUS.Tray_State
Move robot arm	TRAY_STATUS.Tray_Ap
Update data	TRAY_STATUS.Tray_State = TRAY_AP
TRAY POINT	
Verify current state	TRAY_STATUS.Tray_State
Move robot arm	TRAY_HANDLE_LOCATIONS.Working_Loc
Update data	TRAY_STATUS.Tray_State = TRAY
Pick up tray	
Similar to **ACQUIRE**	
Update data	TRAY_STATUS.Current_Tray decremented by 1
TRAY APPROACH POINT	
Verify current state	TRAY_STATUS.Tray_State
Move robot arm	TRAY_STATUS.Tray_Ap
Update data	TRAY_STATUS.Tray_State = TRAY_AP
STORAGE APPROACH POINT	
Verify current state	TRAY_STATUS.Tray_State
Move robot arm	TRAY_STATUS.Storage_Ap
Update data	TRAY_STATUS.Tray_State = STORAGE_AP
STORAGE POINT	
Verify current state	TRAY_STATUS.Tray_State = STORAGE_AP
Move robot arm	TRAY_HANDLE_LOCATIONS.Storage_Loc
Update data	TRAY_STATUS.Tray_State = STOR

```
          Drop tray
            Similar to DROP
          STORAGE APPROACH POINT
            Verify current state         TRAY_STATUS.Tray_State
            Move robot arm               TRAY_STATUS.Storage_Ap
            Update data                  TRAY_STATUS.Tray_State = STORAGE_AP
          TRAY APPROACH POINT
            Verify current state         TRAY_STATUS.Tray_State
            Move robot arm               TRAY_STATUS.Tray_Ap
            Update data                  TRAY_STATUS.Tray_State = TRAY_AP
                                         TRAY_STATUS.Mode = NONE

  3) Exit
ROBOT: (Move robot arm to canister point)
  STRUT POSITION:
    CANISTER APPROACH POINT:
      Verify current state               ROBOT_STATUS.State
      Move robot arm                     STRUT_TYPE.State_Pos
      Update data                        ROBOT_STATUS.State = AP_CAN
  STRUT POSITION:
    CANISTER GRASP POINT:
      Verify current state               ROBOT_STATUS.State
      Move robot arm                     STRUT_TYPE.State_Pos
      Update data                        ROBOT_STATUS.State = GP_CAN
END EFFECTOR: (Pick up strut from tray)
  ACQUIRE:
    Verify strut currently in canister   STRUT_TYPE.Where = CANISTER
    Check latches                        END_EFFECTOR.Latch = LATCHED
    UNLATCH STRUT
    Update data                          END_EFFECTOR.Latch = UNLATCHED
    Check platform                       END_EFFECTOR.Platform = RETRACTED
    EXTEND
    Update data                          END_EFFECTOR.Platform = EXTENDED
    CANISTER BALANCE
    Check latches                        END_EFFECTOR.Latch = UNLATCHED
    LATCH STRUT
    Update data                          END_EFFECTOR.Latch = LATCHED
    CANISTER BALANCE
    Check platforms                      END_EFFECTOR.Platform = EXTENDED
    RETRACT
    Update data                          END_EFFECTOR.Platform = RETRACTED
                                         STRUT_TYPE.Where = ARM
                                         ROBOT_STATUS.Strut_Now = Strutname
                                         ROBOT_STATUS.Getting_now = NONE

ROBOT: (Move robot arm to canister approach
point)
  STRUT POSITION:
    CANISTER APPROACH POINT:
      Verify current state               ROBOT_STATUS.State
      Move robot arm                     STRUT_TYPE.State_Pos
      Update data                        ROBOT_STATUS.State = AP_CAN
```

CONNECT:	
Verify strut currently in robot arm	STRUT_TYPE.where = ARM
Verify robot arm at canister approach point	ROBOT_STATUS.State = AP_CAN
MOTION BASE: (Move motion base to assembly position)	
DEFINED LOCATION:	
PICK LOCATION:	
ASSEMBLY LOCATION:	MOTION_BASE_POSITION.(X_Car, Y_Car, Turntable)
Check current position	CURRENT_MOTION_BASE_POSITION
Perform collision avoidance	STRUT_TYPE.(X_End, Y_End)
Determine execution order	
Move motion base	
Update data	CURRENT_MOTION_BASE_POSITION = MOTION_BASE_POSITION
ROBOT: (Check robot arm position)	
STRUT POSITION:	
Check current state	ROBOT_STATUS.State
END EFFECTOR:	
COMPONENT COMMANDS:	
(Open receptacle fingers)	
Verify receptacle finger status	END_EFFECTOR.Left_Receptacle_Finger = CLOSED
OPEN:	
LEFT RECEPTACLE FINGER	
Update data	END_EFFECTOR.Left_Receptacle_Finger = OPENED
Verify receptacle finger status	END_EFFECTOR.Right_Receptacle_Finger = CLOSED
OPEN:	
RIGHT RECEPTACLE FINGER	
Update data	END_EFFECTOR.Right_Receptacle_Finger = OPENED
ROBOT: (Move robot arm to grasp point from canister approach point)	
STRUT POSITION:	
TRANSITION POINT:	
Verify current state	ROBOT_STATUS.State
Move robot arm	STRUT_TYPE.State_Pos
Update data	ROBOT_STATUS.State = IP
APPROACH POINT:	
Verify current state	ROBOT_STATUS.State
Move robot arm	STRUT_TYPE.State_Pos
Update data	ROBOT_STATUS.State = AP
GRASP POINT:	
Verify current state	ROBOT_STATUS.State
Move robot arm	STRUT_TYPE.State_Pos
Update data	ROBOT_STATUS.State = GP

```
END EFFECTOR: (Install strut into structure)
 INSTALL:
   Verify strut currently in robot arm      STRUT_TYPE.Where = ARM
   Verify receptacle finger status          END_EFFECTOR.Left_Receptacle_Finger =
                                              OPENED
   CLOSE:
     LEFT RECEPTACLE FINGER
     Update data                            END_EFFECTOR.Left_Receptacle_Finger =
                                              CLOSED
   Verify receptacle finger status          END_EFFECTOR.Right_Receptacle_Finger =
                                              OPENED
   CLOSE:
     RIGHT RECEPTACLE FINGER
     Update data                            END_EFFECTOR.Right_Receptacle_Finger =
                                              CLOSED
   BALANCE FTS (Balance the force and
   torques)
   EXTEND
     Update data                            END_EFFECTOR.Platform = EXTENDED
   LOCK:
     LEFT NUT
       Verify left nut status               CURRENT_STRUT.Left_Nut = UNLOCKED
       Put socket over nut
       Verify seating of nut
       Update data                          CURRENT_STRUT.Left_Seat = SEATED
       Lock nut
       Update data                          CURRENT_STRUT.Left_Nut = LOCKED
   LOCK:
     RIGHT NUT
       Verify right nut status              CURRENT_STRUT.Right_Nut = UNLOCKED
       Put socket over nut
       Verify seating of nut
       Update data                          CURRENT_STRUT.Right_Seat = SEATED
       Lock nut
       Update data                          CURRENT_STRUT.Right_Nut = LOCKED
   Check latches                            END_EFFECTOR.Latch = LATCHED
   UNLATCH STRUT
     Update data                            END_EFFECTOR.Latch = UNLATCHED
   Check platforms                          END_EFFECTOR.Platform = EXTENDED
   RETRACT
     Update data                            END_EFFECTOR.Platform = RETRACTED
   Verify receptacle finger status          END_EFFECTOR.Left_Receptacle_Finger =
                                              CLOSED
   OPEN:
     LEFT RECEPTACLE FINGER
       Update data                          END_EFFECTOR.Left_Receptacle_Finger =
                                              OPENED
   Verify receptacle finger status          END_EFFECTOR.Right_Receptacle_Finger =
                                              CLOSED
```

OPEN:	
RIGHT RECEPTACLE FINGER	
Update data	END_EFFECTOR.Right_Receptacle_Finger = OPENED
ROBOT: (Move robot arm to canister approach point from grasp point)	
STRUT POSITION:	
Check current state	
APPROACH POINT:	
Verify current state	ROBOT_STATUS.State
Move robot arm	STRUT_TYPE.State_Pos
Update data	ROBOT_STATUS.State = AP
TRANSITION POINT:	
Verify current state	ROBOT_STATUS.State
Move robot arm	STRUT_TYPE.State_Pos
Update data	ROBOT_STATUS.State = IP
Input the strut name to fetch	
Check data to verify valid strut name	
Verify necessary strut installed	
Update data	
END EFFECTOR: (Close receptacle fingers of all ends with no nodes)	
COMPONENT COMMANDS:	
Close side_1 receptacle finger	
Verify receptacle finger status	END_EFFECTOR.Side_1_Receptacle_Finger = OPENED
CLOSE:	
SIDE_1 RECEPTACLE FINGER	
Update data	END_EFFECTOR.Side_1_Receptacle_Finger = CLOSED
Close side_2 receptacle finger	
Verify receptacle finger status	END_EFFECTOR.Side_2_Receptacle_Finger = OPENED
CLOSE:	
SIDE_2 RECEPTACLE FINGER	
Update data	END_EFFECTOR.Side_2_Receptacle_Finger = CLOSED
ROBOT: (Continue robot arm move to canister approach point)	
STRUT POSITION:	
CANISTER APPROACH POINT:	
Verify current state	ROBOT_STATUS.State
Move robot arm	STRUT_TYPE.State_Pos
Update data	ROBOT_STATUS.State = AP_CAN

Appendix D

Comparison With NASREM

Although the automated-assembly system software is developed from the system requirements, the resulting program structure closely resembles the NASA/NBS Standard Reference Model (NASREM) architecture (ref. 14). The NASREM architecture is depicted in figure 22, and the corresponding automated-assembly software structure is shown in figure 23. The automated-assembly hierarchy corresponds to the four lowest levels of NASREM. For example, the NASREM primitive level can be compared with the automated-assembly device level, which includes the robot arm, the end effector, and the motion base. (See fig. 6.) The NASREM element-move level corresponds to the assembly level in the automated-assembly hierarchy. Figure 23 includes only those functions at each level that are needed in the automated-assembly application. Typical supervisor commands at each level and error-recovery actions are included for completeness.

Hardware actions and sensor processing occur at the component (NASREM servo) level. Error conditions are resolved by either supervisor intervention or automated actions at the component level. Unresolved errors are passed back through the hierarchy, and an automatic reverse of the tasks performed at each level is initiated. For the assembly task, alternative actions, which take the form of substituting other struts for failed members, are available only at the administrative level. A use of alternate struts requires replanning the assembly sequence.

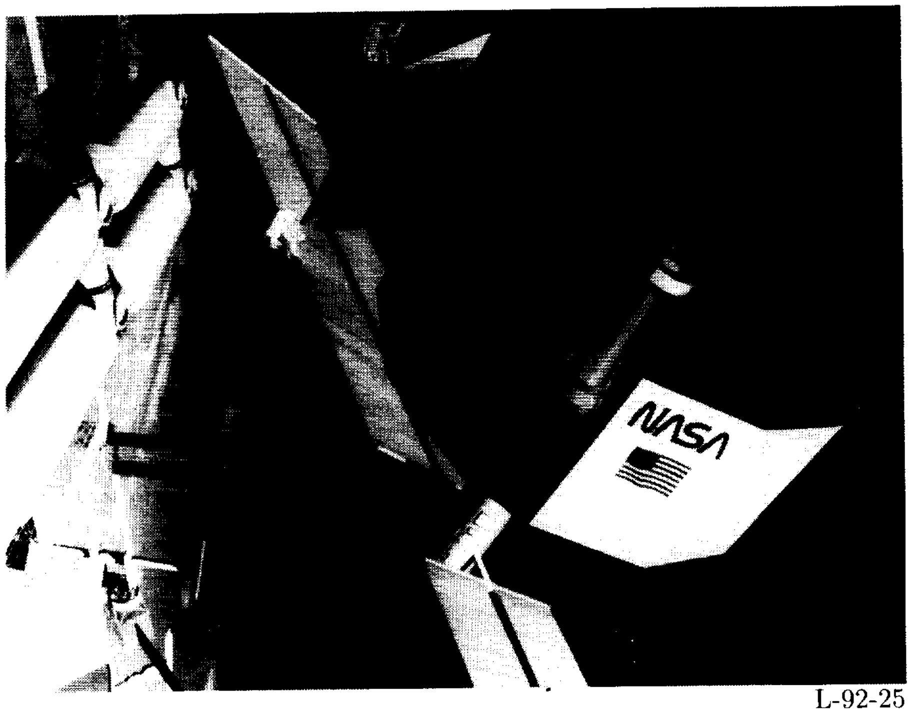
Aside from the component level, the only other testing is performed at the assembly level. These tests involve physically exercising the locking nut immediately after a strut is picked up from the canister to insure that it can be installed. Another test is performed immediately after locking a strut into the structure by attempting to retract the platform before unlatching in order to verify the integrity of the joint lock. A failure of either of these tests would result in the selection of an alternate strut.

The world model information base is updated at two levels--the device level and the assembly level. At the device level, the end-effector status model is updated at the successful completion of each component action. At the assembly level, the truss-structure model and the storage-canister status are updated with the installation or removal of each strut.

The NASREM architecture provides good conceptual agreement with the automated-assembly application, although not all activities have an entry at every level. The hierarchical model does provide a particularly concise display for supervisor visualization. The hierarchical structure is capable of supporting several assembly operations by providing a standard interface between the levels.

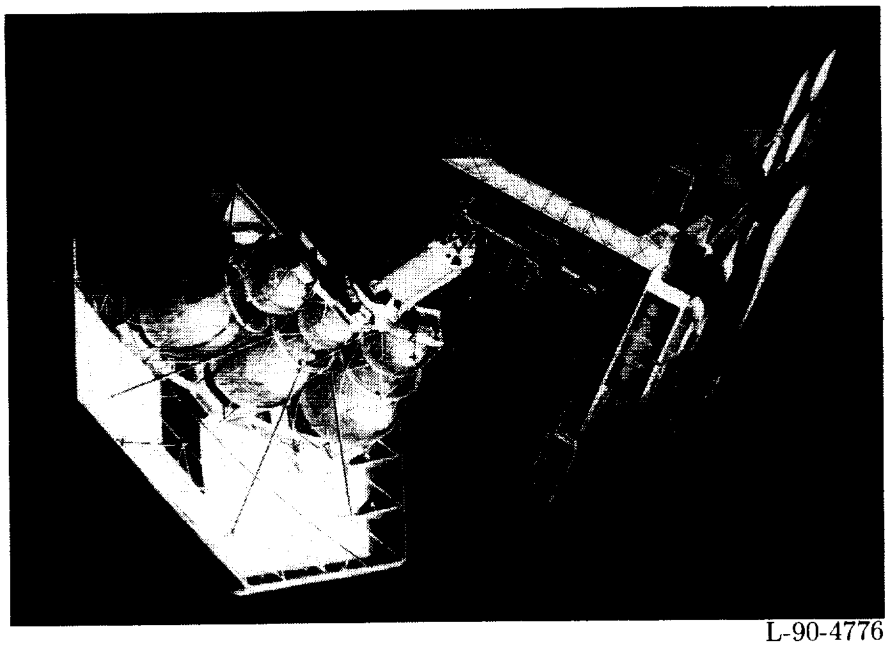
References

1. Heard, Walter L., Jr.; Bush, Harold G.; Wallsom, Richard E.; and Jensen, J. Kermit: *A Mobile Work Station Concept for Mechanically Aided Astronaut Assembly of Large Space Trusses.* NASA TP-2108, 1983.
2. Dorsey, John T.; and Mikulas, Martin M., Jr.: Preliminary Design of a Large Tetrahedral Truss/Hexagonal Panel Aerobrake Structural System. AIAA-90-1050, Apr. 1990.
3. Heard, Walter L., Jr.; Watson, Judith J.; Ross, Jerry L.; Spring, Sherwood C.; and Cleave, Mary L.: Results of the ACCESS Space Construction Shuttle Flight Experiment. *A Collection of Technical Papers—AIAA Space Systems Technology Conference,* American Inst. of Aeronautics and Astronautics, June 1986, pp. 118–125. (Available as AIAA-86-1186.)
4. Bush, Harold G.; Herstrom, Catherine L.; Heard, Walter L., Jr.; Collins, Timothy J.; Fichter, W. B.; Wallsom, Richard E.; and Phelps, James E.: Design and Fabrication of an Erectable Truss for Precision Segmented Reflector Application. *J. Spacecr. & Rockets,* vol. 28, no. 2, Mar.–Apr. 1991, pp. 251–257.
5. *Space Station Freedom External Maintenance Task Team Final Report—Volume I, Part 1.* NASA Lyndon B. Johnson Space Center, July 1990.
6. *Space Station Freedom External Maintenance Task Team Final Report—Volume I, Part 2.* NASA Lyndon B. Johnson Space Center, July 1990.
7. *Space Station Freedom External Maintenance Task Team Final Report—Volume II, Part 1.* NASA Lyndon B. Johnson Space Center, July 1990.
8. *Space Station Freedom External Maintenance Task Team Final Report—Volume II, Part 2.* NASA Lyndon B. Johnson Space Center, July 1990.
9. Rhodes, Marvin D.; Will, Ralph W.; and Wise, Marion A.: *A Telerobotic System for Automated Assembly of Large Space Structures.* NASA TM-101518, 1989.
10. Rhodes, Marvin D.; and Will, Ralph W.: *Automated Assembly of Large Space Structures.* IAF Paper No. 90-272, Oct. 1990.
11. Will, Ralph W.; and Rhodes, Marvin D.: An Automated Assembly System for Large Space Structures. *Cooperative Intelligent Robotics in Space,* Rui J. deFigueiredo and William E. Stoney, eds., Volume 1387 of *Proceedings of SPIE—The International Society for Optical Engineering,* Soc. of Photo-Optical Instrumentation Engineers, 1990, pp. 60–71.
12. *Knowledge Base Author's Manual—KES PS.* Software Architecture & Engineering, Inc., c.1990.
13. Doggett, William R.; Rhodes, Marvin D.; Wise, Marion A.; and Armistead, Maurice F.: A Smart End-Effector for Assembly of Space Truss Structures. Paper presented at SOAR 91—Fifth Annual Space Operations, Applications, and Research Symposium, Final Program (Houston, Texas), July 9–11, 1991.
14. Albus, James S.; McCain, Harry G.; and Lumia, Ronald: *NASA/NBS Standard Reference Model for Telerobot Control System Architecture* (NASREM). NBS Tech. Note 1235, U.S. Dep. of Commerce, July 1987.



L-92-25

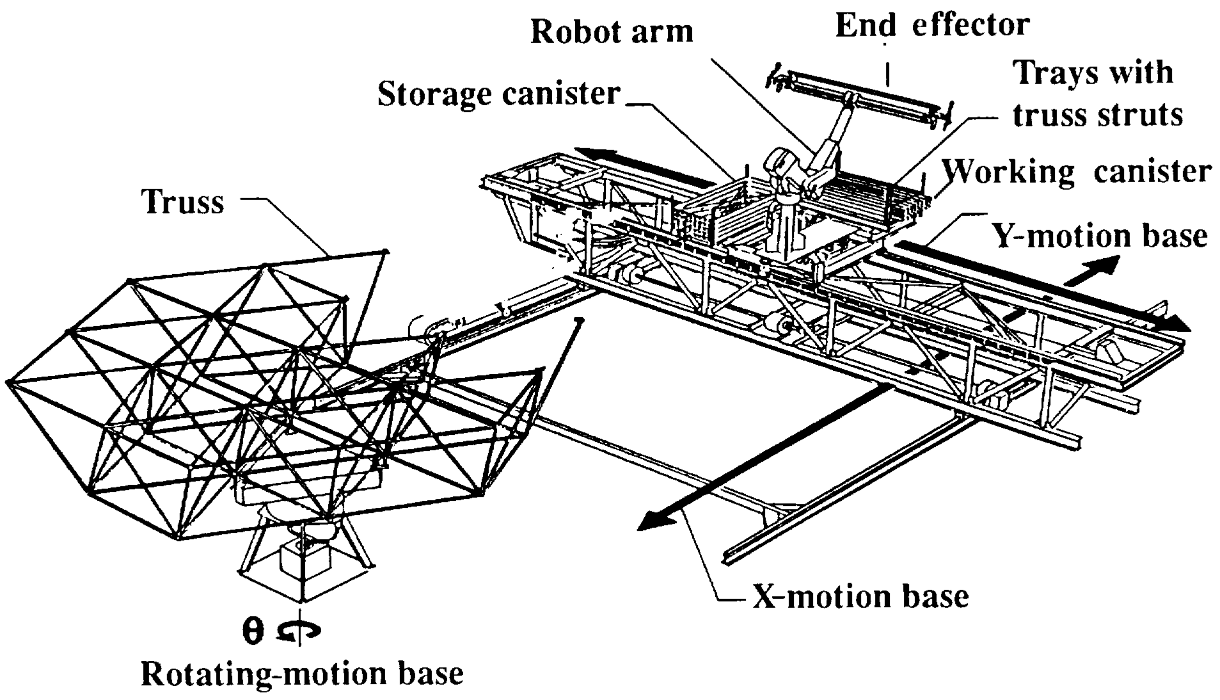
(a) Proposed astronomical observatory.



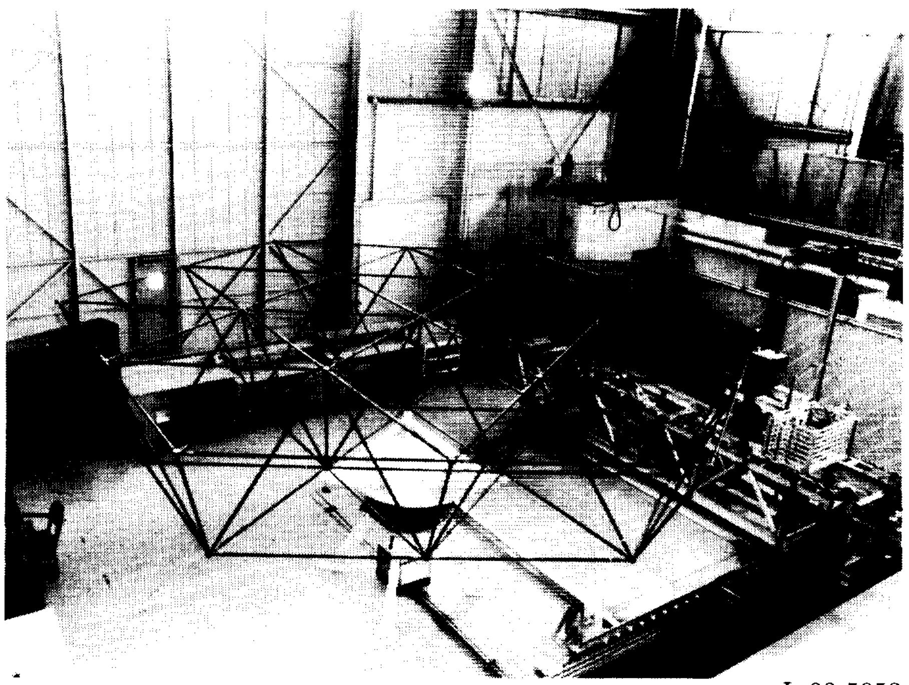
L-90-4776

(b) Concept for a Mars transfer vehicle and aerobrake.

Figure 1. Artist's conception of future space missions.



(a) Schematic.

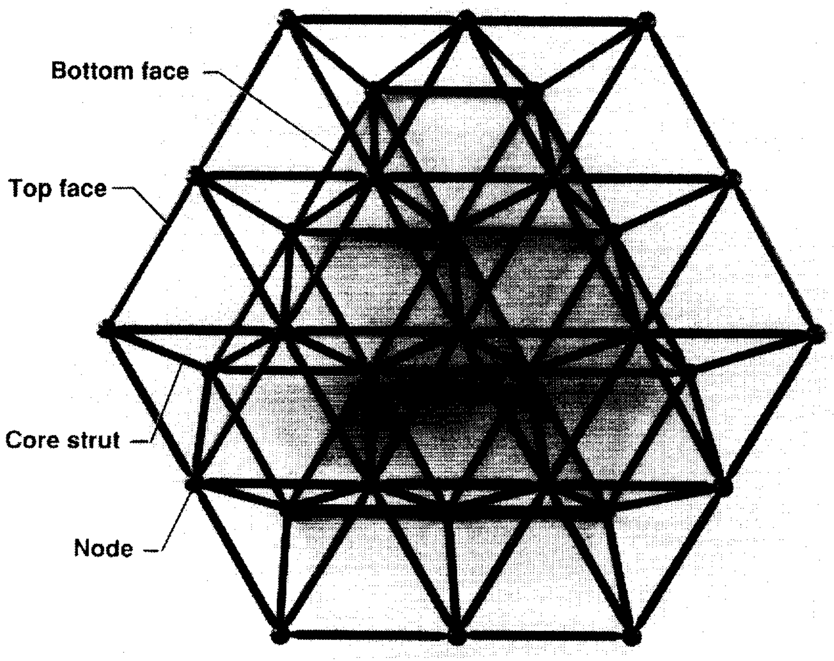


L-90-5053

(b) Photograph.

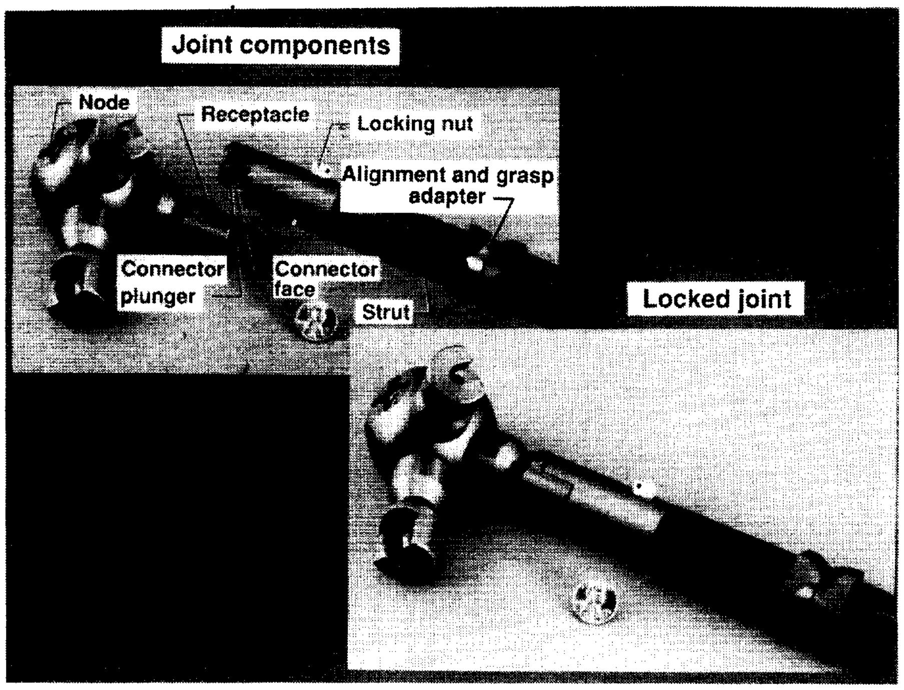
Figure 2. Langley Automated Structure Assembly Laboratory.

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH



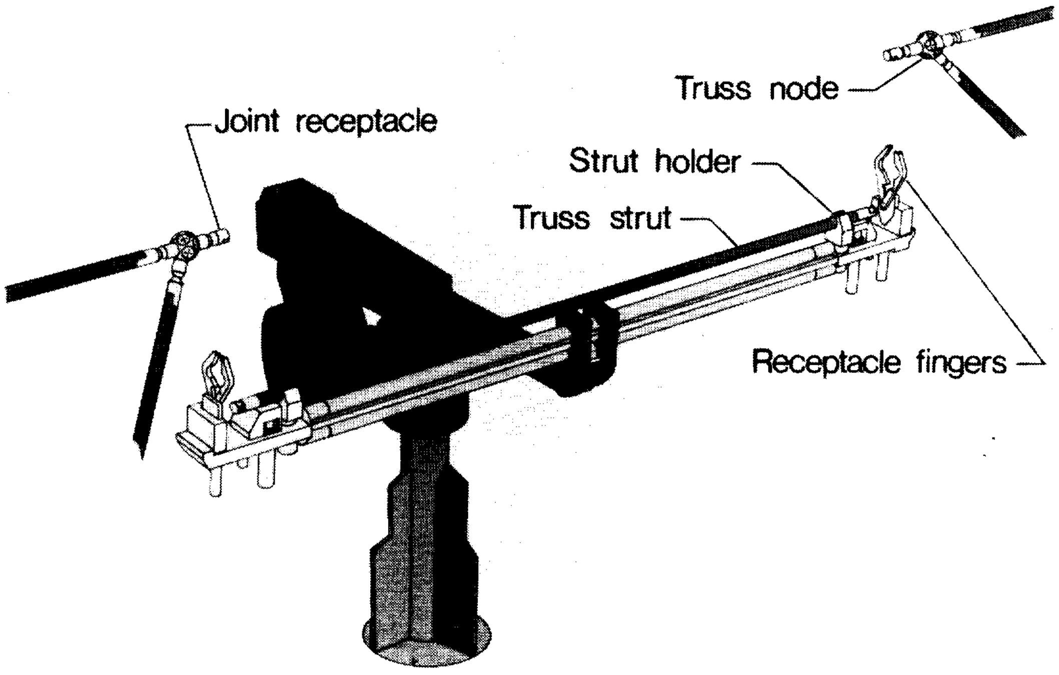
L-88-10,307

Figure 3. Truss geometry.



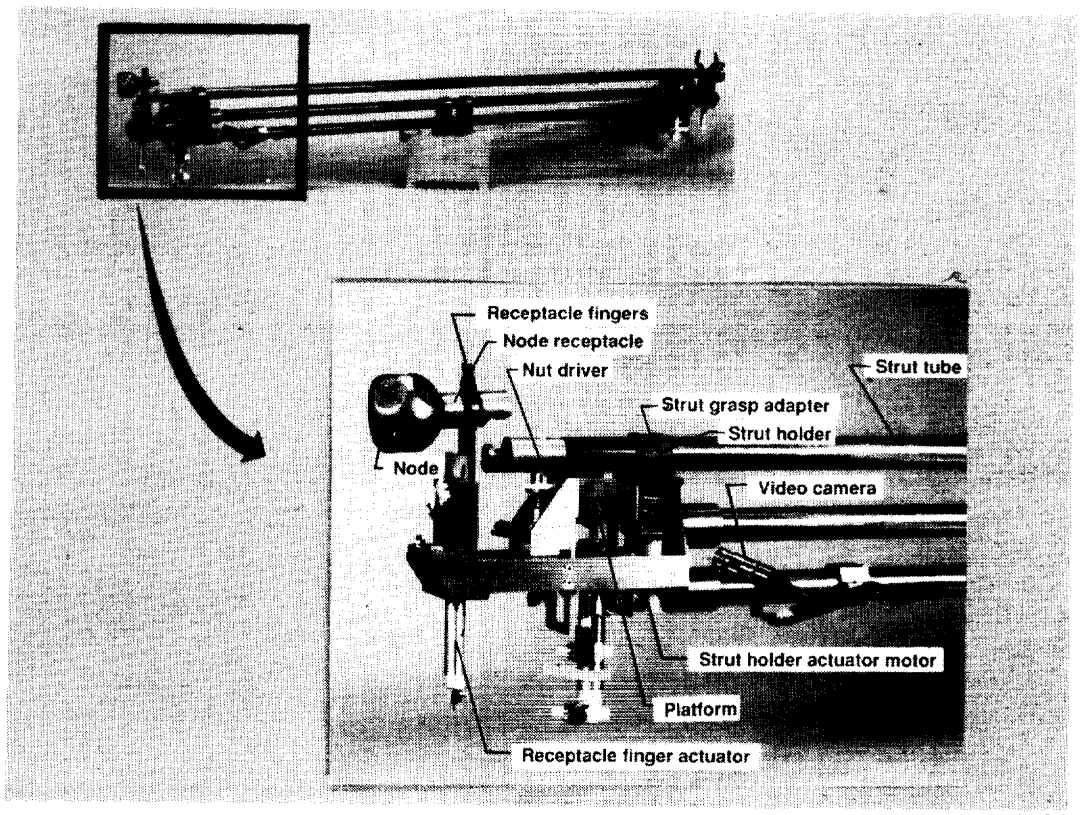
L-90-11104

Figure 4. Truss node and connecting joints.



L-89-4991

(a) Schematic.



L-91-12104

(b) Details of mechanical system.

Figure 5. End-effector tool.

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

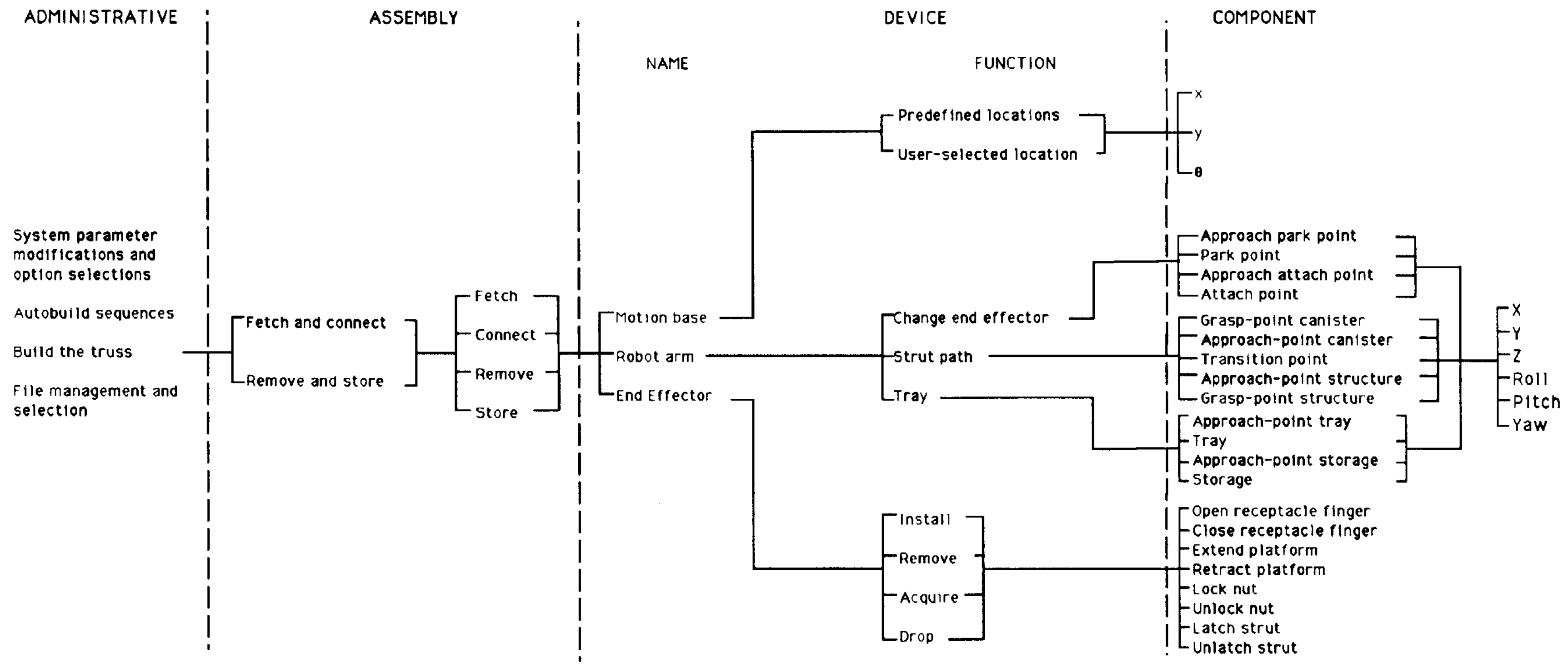


Figure 6. Design layout of automated assembly system.

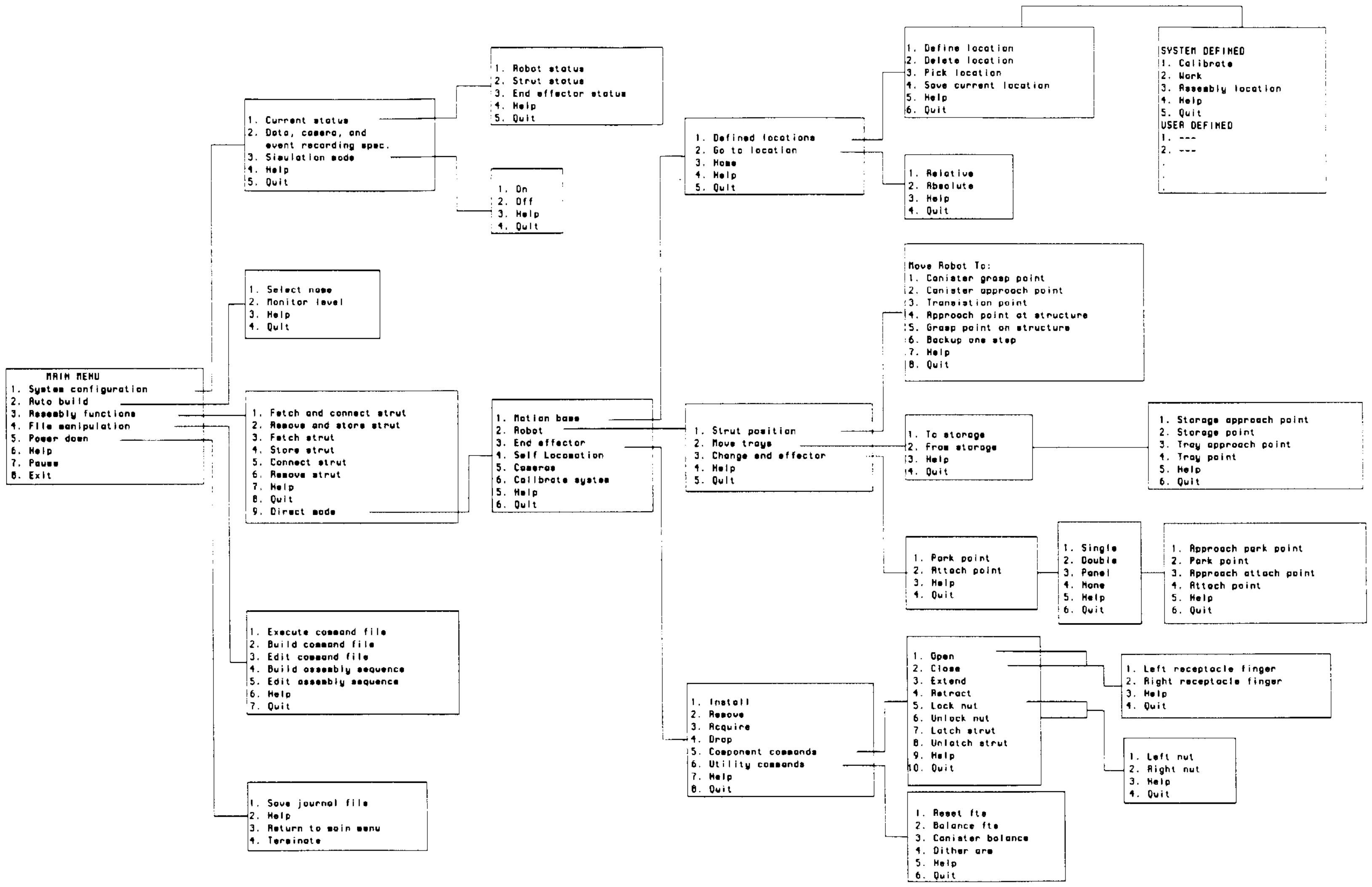
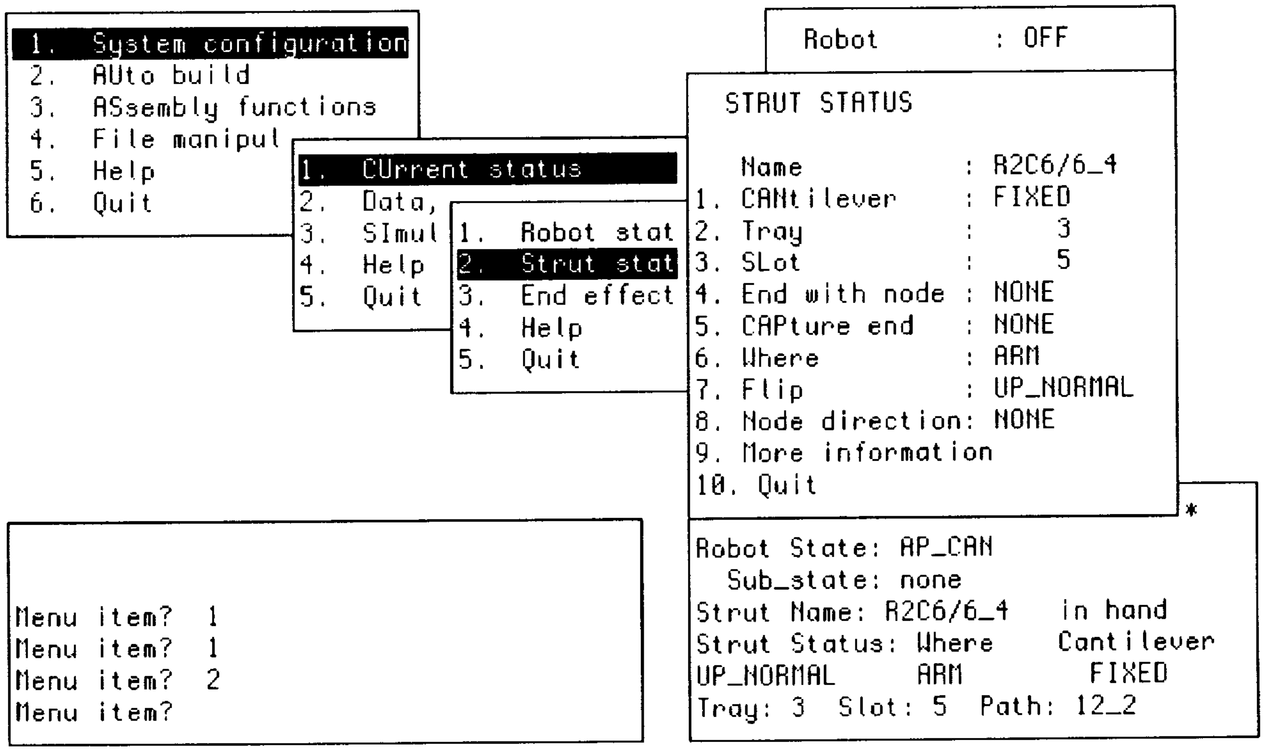


Figure 7. Basic menu layout of automated assembly system software.

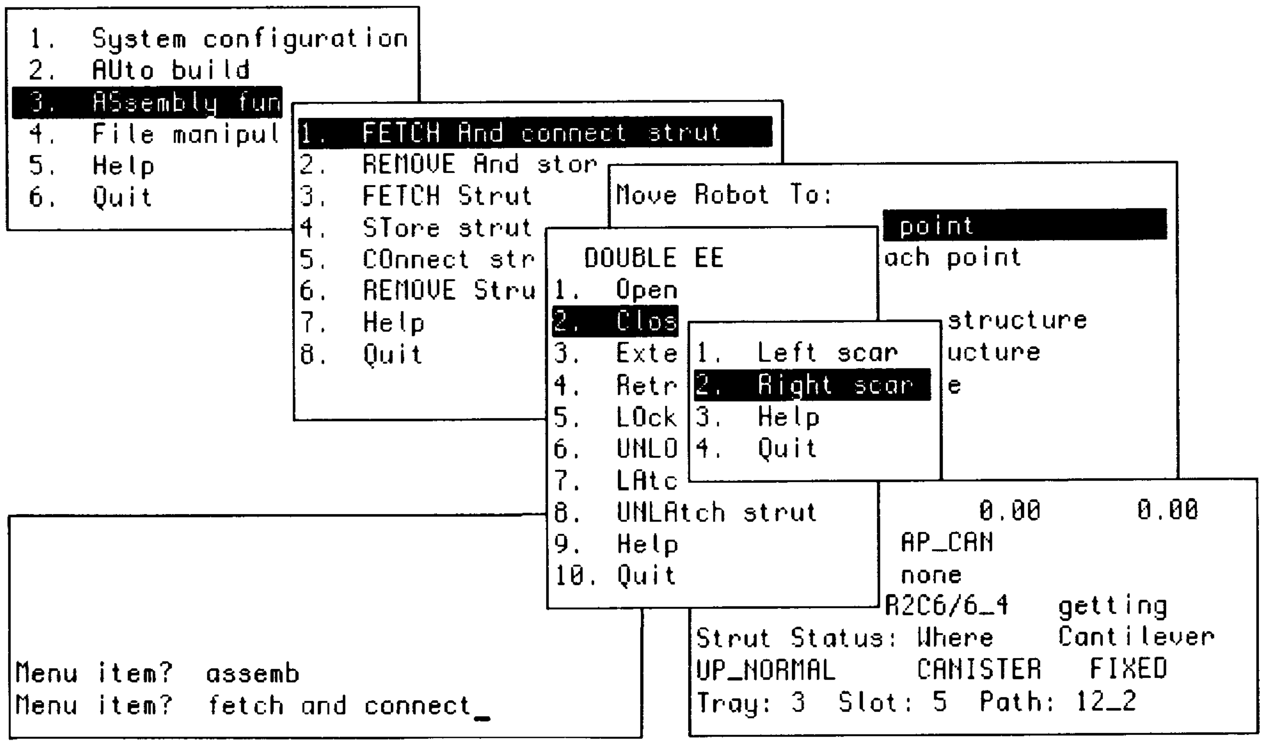


```
Robot          : OFF

STRUT STATUS

   Name              : R2C6/6_4
1. CANtilever        : FIXED
2. Tray              :     3
3. SLot              :     5
4. End with node     : NONE
5. CAPture end       : NONE
6. Where             : ARM
7. Flip              : UP_NORMAL
8. Node direction    : NONE
9. More information
10. Quit
```

```
Menu item?  1
Menu item?  1
Menu item?  2
Menu item?
```

```
Robot State: AP_CAN                          *
  Sub_state: none
Strut Name: R2C6/6_4    in hand
Strut Status: Where     Cantilever
UP_NORMAL       ARM        FIXED
Tray: 3   Slot: 5   Path: 12_2
```

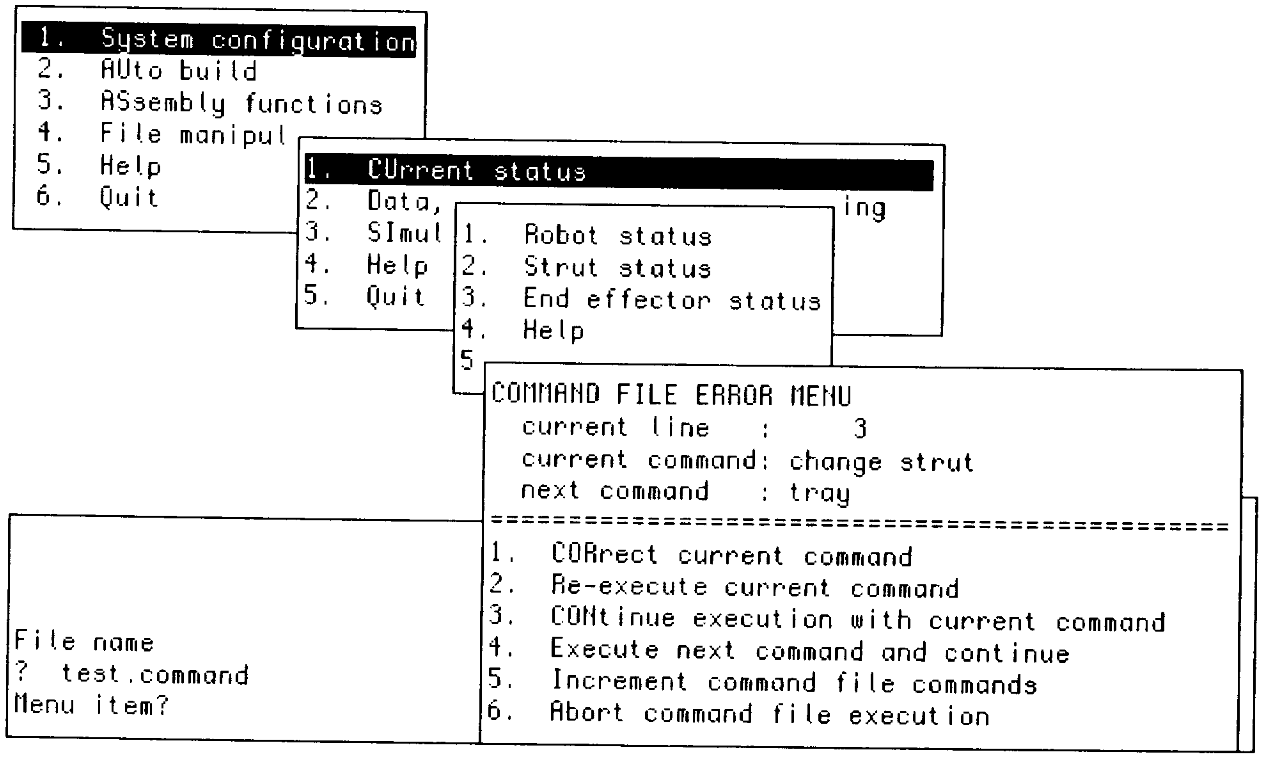
Figure 8. Illustration of hierarchical menu overlay.



```
Menu item?  assemb
Menu item?  fetch and connect_
```

```
                0.00       0.00
 AP_CAN
 none
R2C6/6_4    getting
Strut Status: Where     Cantilever
UP_NORMAL      CANISTER    FIXED
Tray: 3   Slot: 5   Path: 12_2
```

Figure 9. Menu display for automated composite command (fetch and connect).



```
1.  System configuration
2.  AUto build
3.  ASsembly functions
4.  File manipul
5.  Help
6.  Quit

1.  CUrrent status
2.  Data,                          ing
3.  SImul
4.  Help
5.  Quit

1.  Robot status
2.  Strut status
3.  End effector status
4.  Help
5

COMMAND FILE ERROR MENU
  current line    :     3
  current command: change strut
  next command    : tray
=============================================
1.  CORrect current command
2.  Re-execute current command
3.  CONtinue execution with current command
4.  Execute next command and continue
5.  Increment command file commands
6.  Abort command file execution

File name
?  test.command
Menu item?
```

Figure 10. Display of command-file error menu.

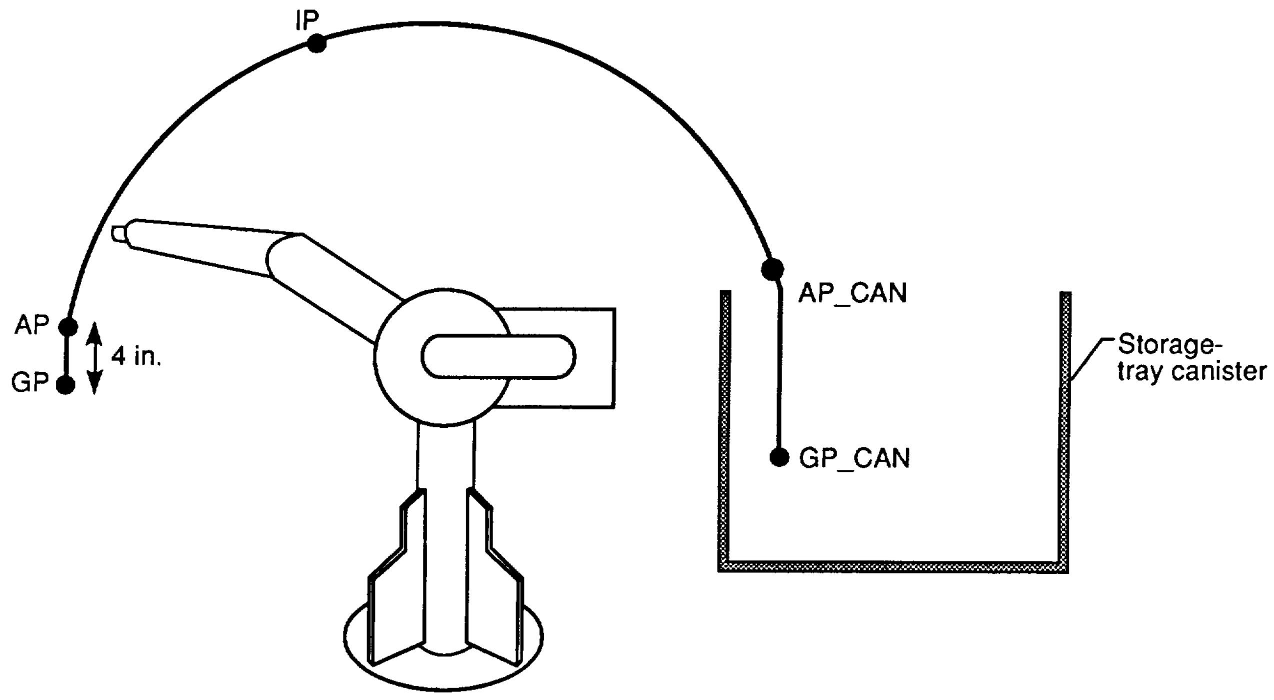


Figure 11. Diagram of robot-arm states for strut paths.

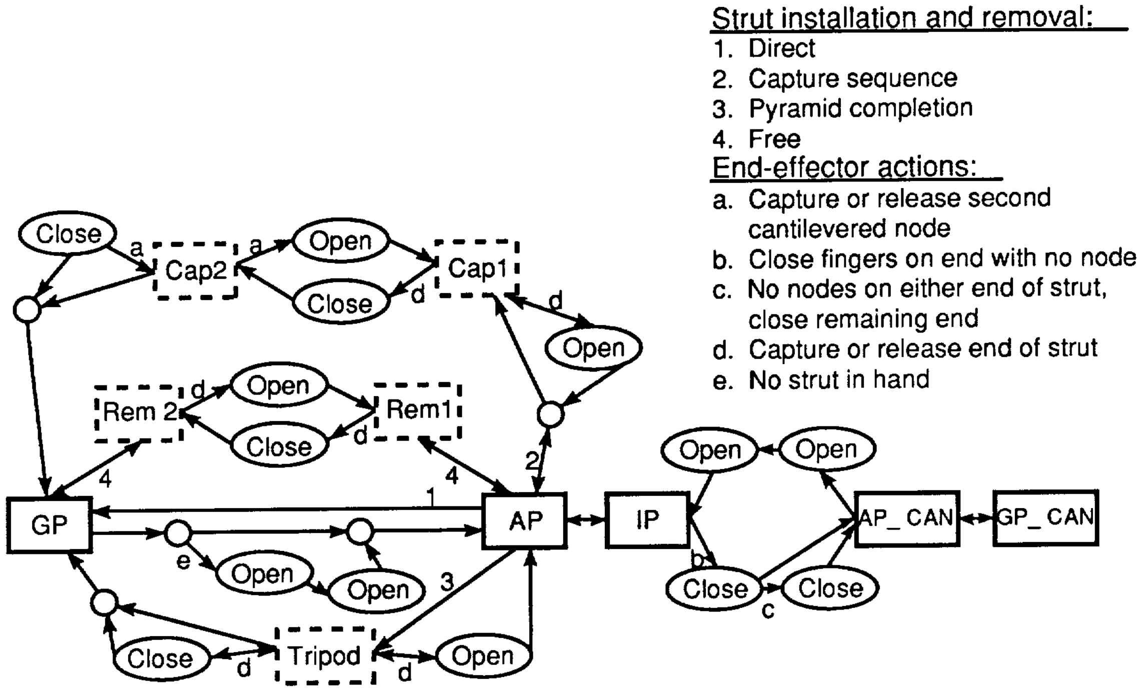
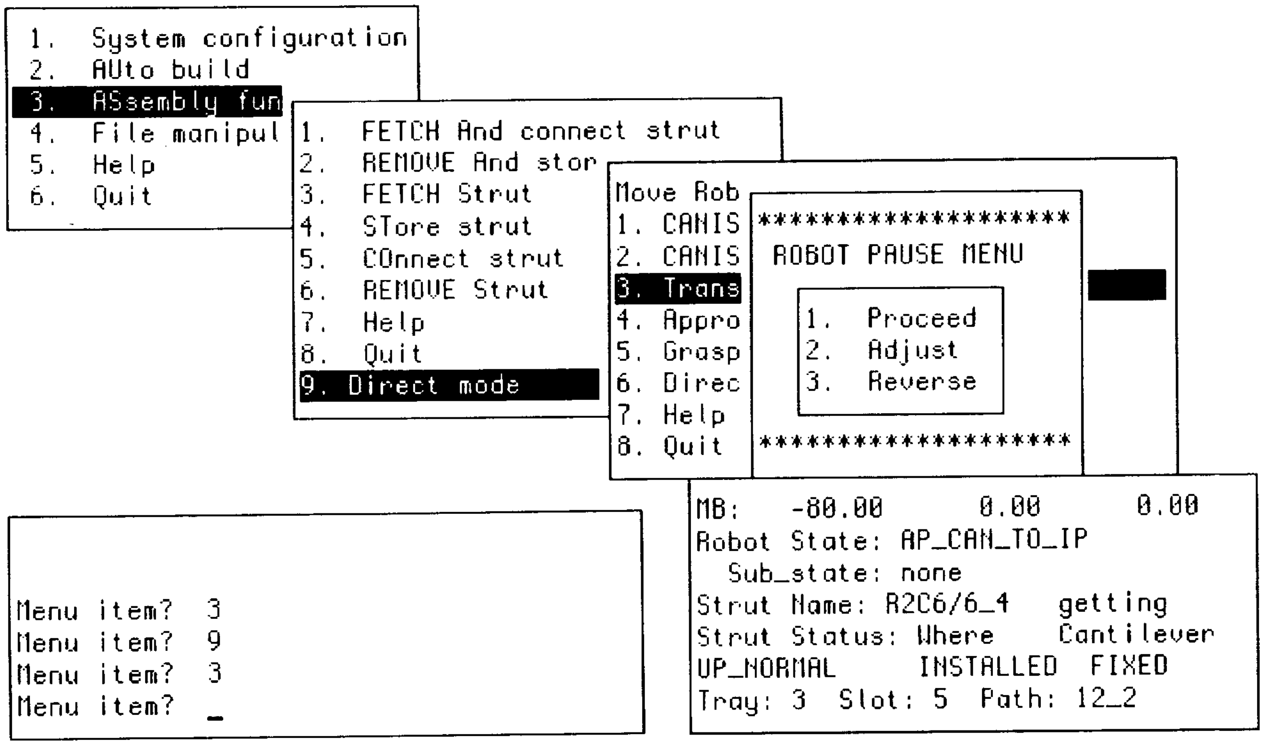

Strut installation and removal:
1. Direct
2. Capture sequence
3. Pyramid completion
4. Free
End-effector actions:
a. Capture or release second cantilevered node
b. Close fingers on end with no node
c. No nodes on either end of strut, close remaining end
d. Capture or release end of strut
e. No strut in hand
Close
Cap2
Open
Close
Cap1
Open
Rem 2
Open
Close
Rem1
GP
AP
IP
Open
Open
Close
Close
AP_CAN
GP_CAN
Open
Open
Close
Tripod
Open

Figure 12. Complete robot-arm state diagram and logic. Dashed boxes indicate conditional states.



```
1.  System configuration
2.  AUto build
3.  ASsembly fun
4.  File manipul
5.  Help
6.  Quit

1.  FETCH And connect strut
2.  REMOVE And stor
3.  FETCH Strut
4.  STore strut
5.  COnnect strut
6.  REMOVE Strut
7.  Help
8.  Quit
9. Direct mode

Move Rob
1. CANIS
2. CANIS
3. Trans
4. Appro
5. Grasp
6. Direc
7. Help
8. Quit

********************
ROBOT PAUSE MENU
 1.  Proceed
 2.  Adjust
 3.  Reverse
********************

Menu item?  3
Menu item?  9
Menu item?  3
Menu item?  _

MB:    -80.00       0.00       0.00
Robot State: AP_CAN_TO_IP
  Sub_state: none
Strut Name: R2C6/6_4    getting
Strut Status: Where     Cantilever
UP_NORMAL     INSTALLED  FIXED
Tray: 3  Slot: 5  Path: 12_2
```

Figure 13. Supervisor display of robot-arm pause menu.

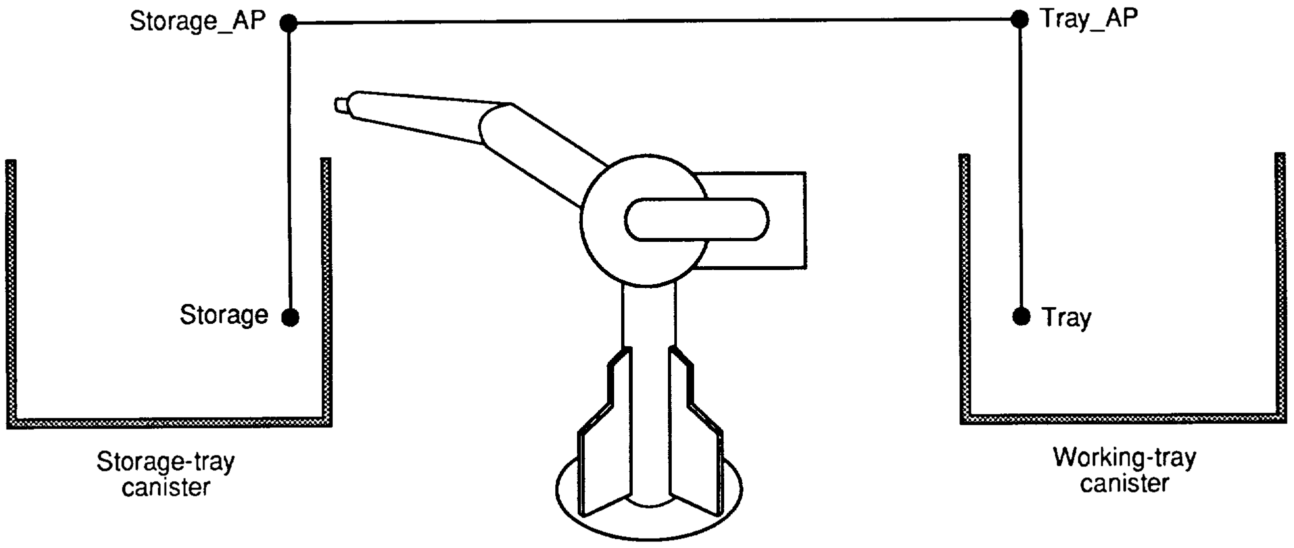


Figure 14. Diagram of robot-arm states for tray moves.

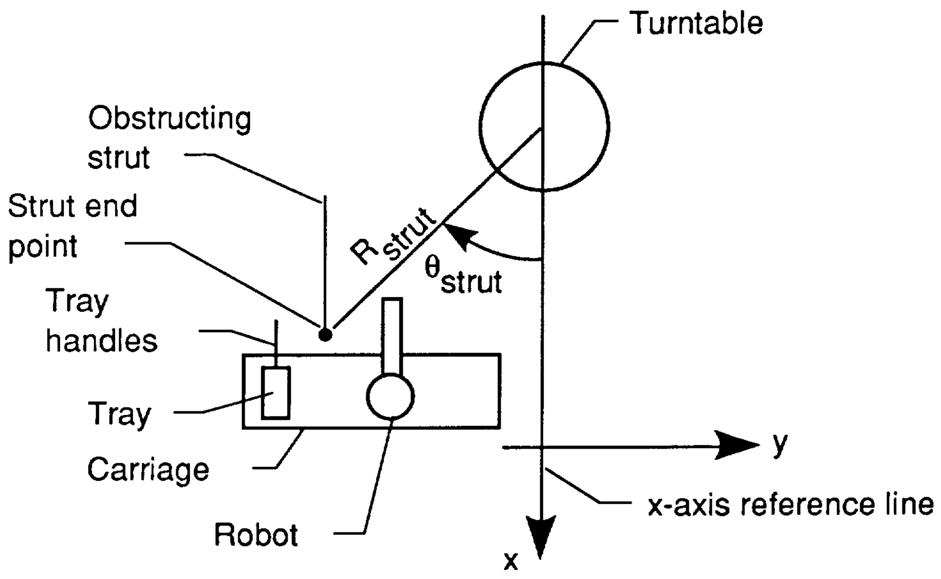


Figure 15. Descriptive diagram of collision avoidance.

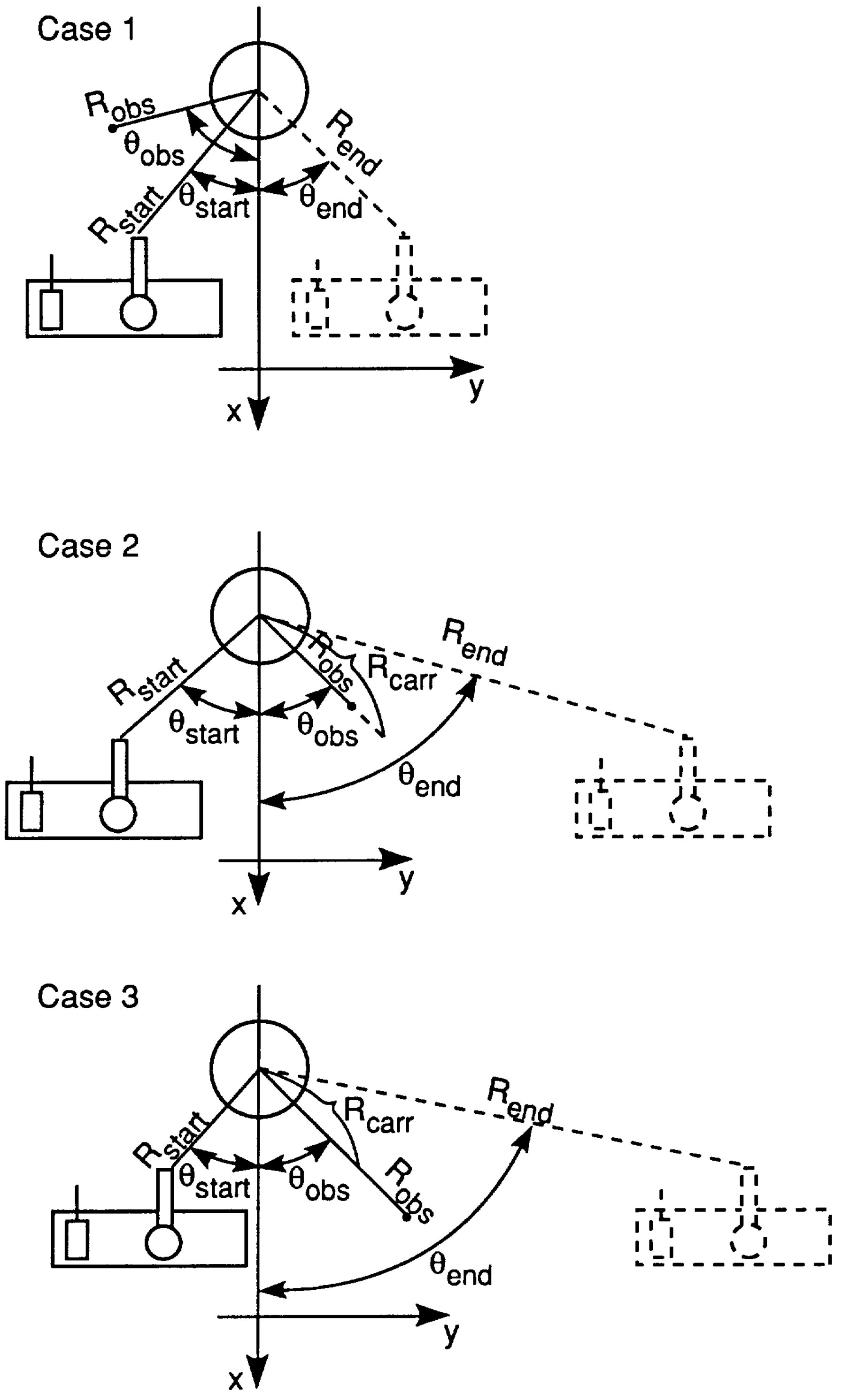


Figure 16. Carriage collision logic for y-axis move. Dashed lines are desired position of carriage.

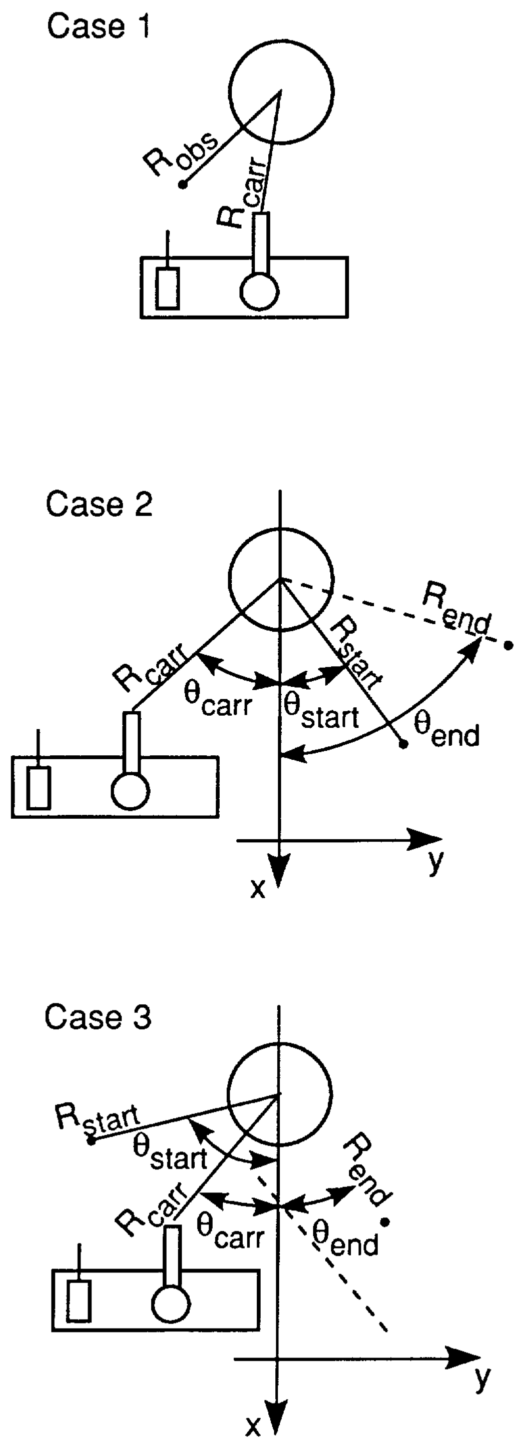
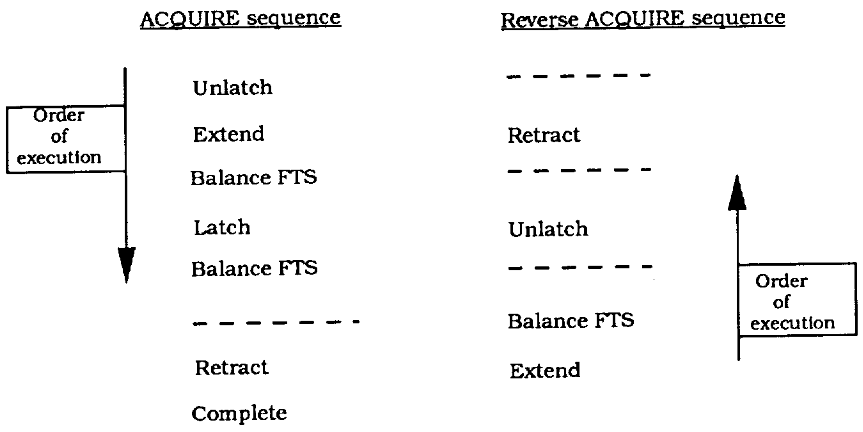
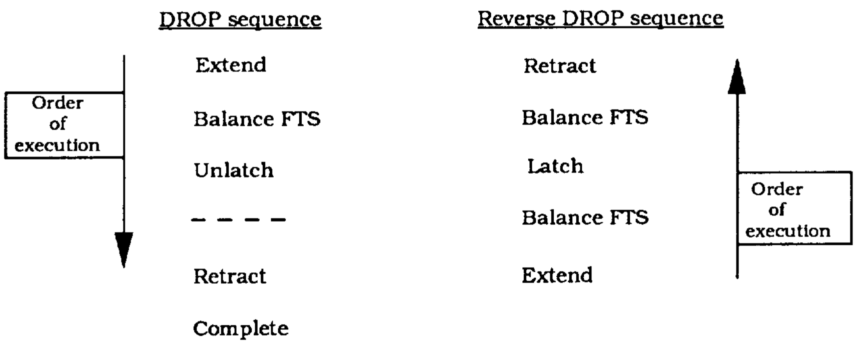


Figure 17. Carriage collision logic for turntable rotation. Dashed lines are desired position of carriage.

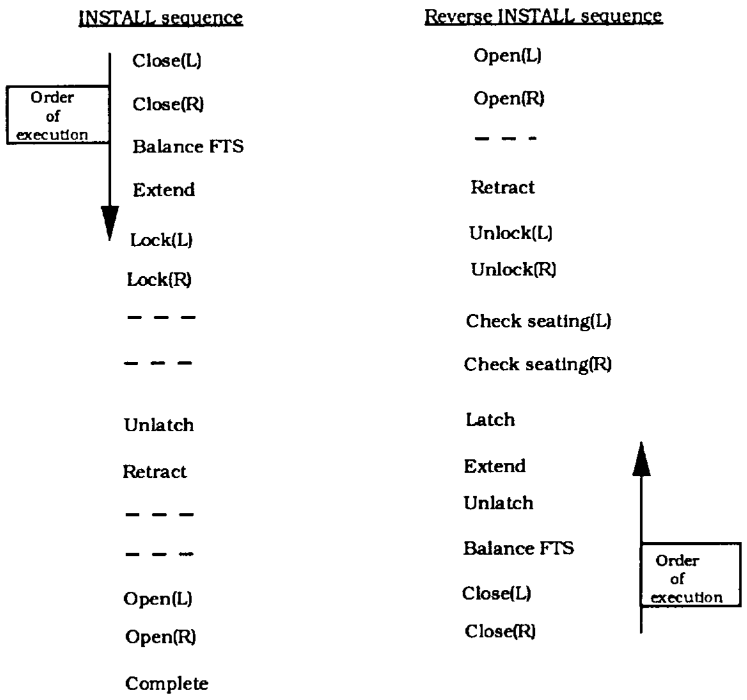


(a) ACQUIRE command.

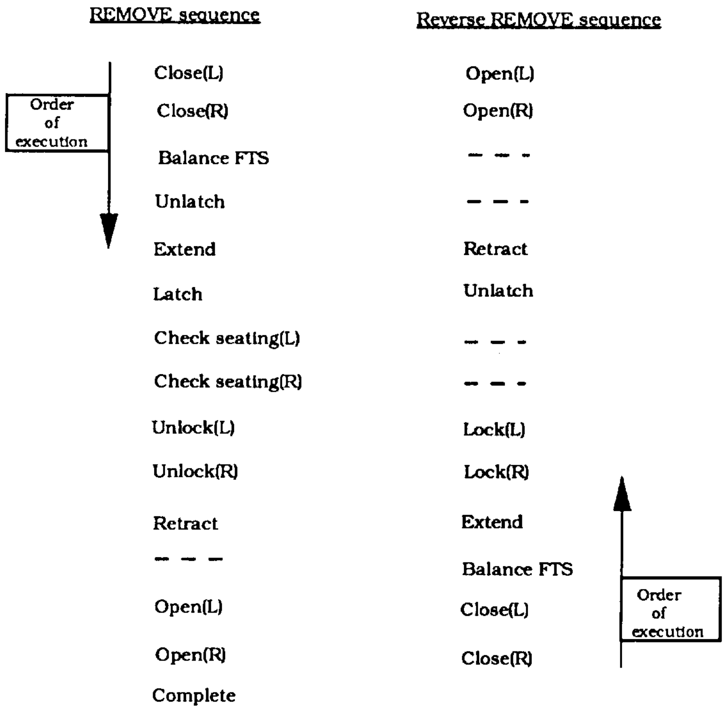


(b) DROP command.

Figure 18. Operational sequences for end-effector assembly functions.

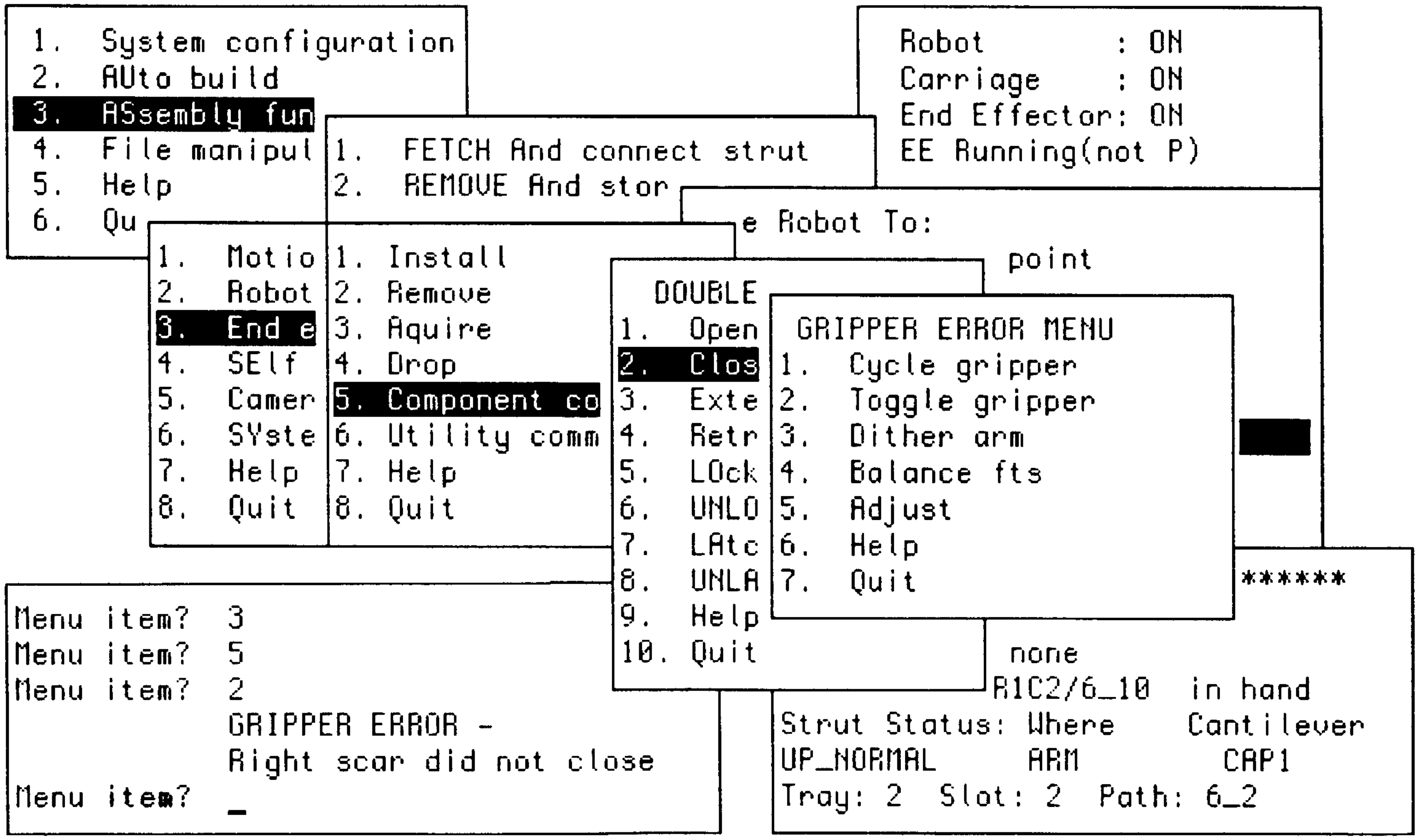


(c) INSTALL command.



(d) REMOVE command.

Figure 18. Concluded.



```
1.  System configuration
2.  AUto build
3.  ASsembly fun
4.  File manipul
5.  Help
6.  Qu
```

```
1.  FETCH And connect strut
2.  REMOVE And stor
```

```
Robot         : ON
Carriage      : ON
End Effector: ON
EE Running(not P)
```

```
e Robot To:
                point
```

```
1.  Motio
2.  Robot
3.  End e
4.  SElf
5.  Camer
6.  SYste
7.  Help
8.  Quit
```

```
1. Install
2. Remove
3. Aquire
4. Drop
5. Component co
6. Utility comm
7. Help
8. Quit
```

```
 DOUBLE
1.  Open
2.  Clos
3.  Exte
4.  Retr
5.  LOck
6.  UNLO
7.  LAtc
8.  UNLA
9.  Help
10. Quit
```

```
GRIPPER ERROR MENU
1.  Cycle gripper
2.  Toggle gripper
3.  Dither arm
4.  Balance fts
5.  Adjust
6.  Help
7.  Quit
```

```
Menu item?  3
Menu item?  5
Menu item?  2
            GRIPPER ERROR -
            Right scar did not close
Menu item?  _
```

```
******

   none
  R1C2/6_10   in hand
Strut Status: Where    Cantilever
UP_NORMAL      ARM        CAP1
Tray: 2  Slot: 2  Path: 6_2
```

Figure 19. End-effector error-recovery menu.

Record	Variable	Description
MOTION_BASE_POSITION		Motion-base configuration
	X_Car	Carriage x position
	Y_Car	Carriage y position
	Turntable	Theta angle in degrees
STRUT_TYPE		Strut description
	Name	Strut name
	Where	Current strut location
	Connect_To	Struts needed for support
	Loc_In_Cell	Installation position in truss
	Node_End	End with node
	Cap_End	End to capture
	Cantilever	End conditions of strut
	Flip	Flip indication
	Tray	Tray number containing strut
	Slot	Position in tray
	State_Pos	Position for each robot state
	X, Y, Z, Roll, Pitch, Yaw	
	X_End, Y_End	Core strut end points for collision avoidance
ROBOT_STATUS		Manipulator arm current mode
	State	Robot path location
	Cond_State	Substate point in a path
	Strut_Now	Name of strut in arm, if any
	Strut_Getting_Now	Name of strut in process of retrieving
	Strut_Just_Had	Name of last strut installed
TRAY_STATUS		Current tray locations and mode
	Tray_State	Path location
	Tray_Mode	Current tray operation
	Current_Tray	Tray on top
	Working_Ap	Robot position for working approach point
	X, Y, Z, Roll, Pitch, Yaw	
	Storage_Ap	Robot position for storage approach point
	X, Y, Z, Roll, Pitch, Yaw	
TRAY_HANDLE_LOCATIONS		
	Storage_Loc	Robot position in storage canister
	X, Y, Z, Roll, Pitch, Yaw	
	Working_Loc	Robot position in working canister
	X, Y, Z, Roll, Pitch, Yaw	
CURRENT_STRUT		Strut currently in arm
	Left_Seat	Nut-driver alignment status
	Right_Seat	
	Left_Nut	Nut status
	Right_Nut	
CURRENT_MOTION_BASE_POSITION		Motion-base configuration
	X_Car	Carriage x position
	Y_Car	Carriage y position
	Turntable	Lazy Susan theta angle in degrees
END_EFFECTOR		End-effector status
	Left_Receptacle_Finger	Left receptacle fingers status
	Right_Receptacle_Finger	Right receptacle fingers status
	Platform	Status of platform
	Latch	Strut holders status
	Storage_Pos	Robot position for storing
	X, Y, Z, Roll, Pitch, Yaw	

Figure 20. Data variables and descriptions.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 1992	3. REPORT TYPE AND DATES COVERED Technical Paper

4. TITLE AND SUBTITLE Software Design for Automated Assembly of Truss Structures	5. FUNDING NUMBERS WU 506-43-41-02
6. AUTHOR(S) Catherine L. Herstrom, Carolyn Grantham, Cheryl L. Allen, William R. Doggett, and Ralph W. Will	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, VA 23665-5225	8. PERFORMING ORGANIZATION REPORT NUMBER L-16983
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA TP-3198
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified–Unlimited Subject Category 63	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Concern over limited extravehicular and intravehicular activitiy time has increased the interest in performing in-space assembly and construction operations with automated robotic systems. A technique being considered at Langley Research Center is a supervised-autonomy approach, which can be monitored by an Earth-based supervisor that intervenes only when the automated system encounters a problem. A test-bed to support evaluation of the hardware and software requirements for supervised-autonomy assembly methods has been developed. This report describes the design of the software system necessary to support the assembly process. The system is implemented and successfully assembles and disassembles a planar tetrahedral truss structure. The software is hierarchical and supports both automated assembly operations and supervisor error-recovery procedures, including the capability to pause and reverse any operation. The software design serves as a model for the development of software for more sophisticated automated systems and as a test-bed for evaluation of new concepts and hardware components.

14. SUBJECT TERMS Robotic; Assembly of truss structures; Automated assembly; Software design			15. NUMBER OF PAGES 45
			16. PRICE CODE A03
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT